On the weakness of conditional equations in algebraic specification

Arno Schönegge* and David Kempe**

*Xcc Software AG, D-76137 Karlsruhe, Germany
Arno.Schoenegge@xcc.de

**Department of Computer Science, Cornell University, Ithaca, NY 14853
kempe@cs.cornell.edu

Abstract. In the initial semantics approach to algebraic specification, one usually restricts oneself to conditional equations. It is well-known that this restriction does not affect expressiveness, at least if the introduction of auxiliary (hidden) functions is allowed: all the computable data types can be specified using conditional equations and initial algebra semantics [BT82].
However, the situation is quite different concerning the specification *without* hidden functions. In this case, the completeness for computable data types is lost for the initial specification methods (cf. [Maj79, TWW82, BT87]) as well as for the loose ones (cf. [Ber93, KS98]). Instead of considering these two approaches separately, it is natural to ask how their expressive powers compare to each other. It is already known that the loose specification methods (even allowing full first-order logic) are not more powerful than the initial methods (even using equations only) [Sch97]. In this paper we give the answer to the opposite question. More precisely, we provide an example of a computable data type which has a monomorphic quantifier-free specification under loose semantics and prove that it fails to possess a conditional equational specification under initial semantics (both without hidden functions). Thus, our main result is that, without hidden functions, the restriction to conditional equations *does* affect expressiveness—even if compared to quantifier-free axioms.

1 Introduction

Most algebraic specification languages can be roughly classified as belonging to either the initial or the loose approach (see [Wir95] or [SW98] for an overview).[1]

In the initial approach, the semantics of a specification is defined to be the initial[2] model of the axioms, i.e., it is characterized by "no junk" (each data item can be constructed using only the constants and functions in the signature) and "no confusion" (two data items are equivalent iff they can be proved equal from the axioms) [GTWW75, EM85]. In order to guarantee executability and the existence of initial models, one usually restricts the axioms to conditional equations

[1] Several alternative approaches have been proposed (e.g. final algebra semantics [Wan81]), but are of less practical relevance.
[2] We give precise definitions in section 2.

or even equations only. Some examples of specification languages following the initial approach are `Clear`, `OBJ`, and `ACT-ONE`.

In the loose approach the semantics of a specification is defined to be the set of all term-generated models of the axioms [GGM76, WPP+83]. Usually, full first-order logic or its quantifier-free subset can be used for the axioms. Examples of specification languages following the loose approach are `ASL` and the `Larch` shared language.

A problem that immediately arises is to investigate the expressive power of the various approaches. In fact, this question has been studied over several years. One fundamental result is that, if equipped with hiding mechanisms, all common algebraic specification methods are adequate for computable data types [BT82, BT87]. Here, hiding means that local definitions of auxiliary functions can be added for reasons of specification only; these functions are hidden from the user of the specified data type.

However, if we do not allow hidden functions—as we suppose throughout this paper—the situation changes substantially. Majster [Maj79] and Bergstra & Tucker [BT87, THEOREM 4.8] gave examples of computable data types which cannot be specified using conditional equations and initial algebra semantics. Orejas [Ore79], Thatcher, Wagner & Wright [TWW82, THEOREM 5], and also Bergstra & Meyer [BM84] proved that conditional equations are strictly more powerful than equations.

Concerning the loose approach, the corresponding questions have been answered only recently. Berghammer [Ber93] showed that the computable data type $(\mathbb{Z};\ 0,\ x+1,\ x-1)$ of integers cannot be monomorphically specified with first-order formulas under loose semantics (cf. also [Sch97]). An example of a computable data type which demonstrates that full first-order axioms are strictly more powerful than quantifier-free formulas was provided in [KS98].

Thus, the relevant questions concerning expressiveness without hidden functions are answered for both the initial and the loose approach. However, instead of considering these methods separately, it is natural to oppose the initial methods to the loose ones. This results in two questions:

- Is the loose approach more powerful than the initial approach?
- Is the initial approach more powerful than the loose approach?

The answer to the first question is already known to be "no": the data type of integers mentioned above does have an equational specification under initial semantics. However, to our knowledge, there has been no satisfactory answer to the second question. On the one hand, it is known that quantifiers increase expressiveness: the data type (of completeness of finite graphs) given in [KS98] can be specified under loose semantics using first-order axioms but not without quantifiers. In particular, it cannot be specified under initial semantics using conditional equations. The remaining problem deals with the case that quantifiers are dispensed with even in the loose approach. This paper is intended to close the gap. We are going to prove the following:

Theorem 1. *There is a computable data type that can be (monomorphically) specified under loose semantics using quantifier-free axioms, but not under initial semantics using conditional equations (both without hidden functions).*

This result is, although very much in the style of the results mentioned above, less obvious. At first sight, one might suspect the opposite—as we will discuss in section 3. The proof of the theorem is done constructively by providing a suitable data type (section 4) and proving its specifiability with loose methods (section 5) and its non-specifiability with initial methods (section 6). First, we recall some basic definitions.

2 Preliminaries

We assume the reader to be familiar with the very basic notions of algebraic specification (cf. e.g. [EM85, Wir90, LEW96]) like those of (many-sorted) *signature* $\Sigma = (S, F)$, (total) *Σ-algebra* $A = ((s^A)_{s \in S}, (f^A)_{f \in F})$, and *$\Sigma$-homomorphism*.

The set of *terms* $T(\Sigma, X)$ with variables taken from X is defined as usual. Terms without variables are called *ground terms*. A Σ-algebra A is called *term-generated* if for each of its carrier elements $a \in s^A$ $(s \in S)$, there is a denotation, i.e. a ground term $t \in T(\Sigma, \emptyset)$ with $t^A = a$. The class of all term-generated Σ-algebras is denoted by $Gen(\Sigma)$.

Two Σ-algebras A, B are called *isomorphic* if there is a bijective Σ-homomorphism $h : A \to B$. The isomorphism class of a Σ-algebra A, also called the "abstract" *data type*, is denoted by $[A]$. Sometimes, we simply speak of the *data type* A. For a class C of Σ-algebras, $A \in C$ is called *initial in* C if for all $B \in C$ there exists a unique Σ-homomorphism $h : A \to B$.

A Σ-algebra is called *computable* if it is isomorphic to a *computable number algebra*, an algebra in which all carrier sets are decidable subsets of $\mathbb{N}$ and all functions are computable.

Atomic Σ-formulas are Σ-equations $t_1 = t_2$ (where $t_1, t_2 \in T(\Sigma, X)$ are terms of the same sort) and the boolean constant **false**. *(First-order) Σ-formulas* are built from the atomic ones with the logical connectives $\neg$ and $\wedge$, and the quantifier $\exists$. Further logical operators such as **true**, $\neq$, $\vee$, $\rightarrow$, $\leftrightarrow$, and $\forall$ are regarded as abbreviations. *Conditional equations* are formulas of the form:

$$t_1 = t_1' \ \wedge \cdots \wedge \ t_k = t_k' \ \rightarrow \ t = t'.$$

The usual *satisfaction* of a Σ-formula φ by a Σ-algebra A w.r.t. a valuation $v : X \to A$ is denoted by $A, v \models \varphi$. We write $A \models \varphi$, and say that φ is *valid in* A, if $A, v \models \varphi$ holds for all valuations v. For a set Φ of Σ-formulas, we write $A \models \Phi$ if $A \models \varphi$ for all $\varphi \in \Phi$.

A *(first-order algebraic) specification* $SP = (\Sigma, \Phi)$ consists of a signature Σ and a finite set Φ of Σ-formulas, called *axioms*. Two semantic functions are defined corresponding to the two main approaches:

- *loose semantics*: $Mod(SP) := \{A \in Gen(\Sigma) \mid A \models \Phi\}$
- *initial algebra semantics*:[3] $I(SP) := \{A \in Mod(SP) \mid A \text{ initial in } Mod(SP)\}$.

The elements of $Mod(SP)$ and $I(SP)$ are called the *models* and *initial models*, respectively. A specification $SP = (\Sigma, \Phi)$ is said to *specify* a Σ-algebra A (and also the corresponding abstract data type $[A]$)

- *under loose semantics* if $Mod(SP) = [A]$
- *under initial algebra semantics* if $I(SP) = [A]$.

A specification SP is called *monomorphic* if any two of its models are isomorphic, i.e., if there is—up to isomorphism—at most one element in $Mod(SP)$.[4] In particular, a specification SP specifies a Σ-algebra A under loose semantics if and only if SP is monomorphic and $A \in Mod(SP)$.

While in the initial algebra approach, one usually restricts oneself to conditional equations (in order to guarantee executability and the existence of initial models), in the loose semantics approach quantifier-free or arbitrary first-order formulas are used as axioms.

A Σ-algebra A with $\Sigma = (S, F)$ can be specified *with hidden functions* (under loose or initial algebra semantics) if there is a super-signature $\Sigma' = (S, F \cup HF)$ (where HF denotes the additional hidden function symbols) and a Σ'-algebra A' with A as its Σ-reduct, i.e. $A'|_\Sigma = A$, such that A' possesses a specification (under loose or initial algebra semantics, respectively).

3 Discussion of the problem

The aim is to prove that there is a computable data type that can be specified using quantifier-free loose methods but not using the conditional initial approach (both without hidden functions).

This claim is very much in the style of the results mentioned in the introduction (e.g. [TWW82, THEOREM 2 and 5], [Maj79, LEMMA 6], [BM84], and [BT87, THEOREM 4.8]). Nevertheless, it is somewhat surprising—as we are going to discuss in this section.

First, remember that any finite set of quantifier-free formulas can be rewritten in conjunctive normal-form preserving equivalence and thus into a finite set of formulas of the form:

$$t_1 = t_1' \ \wedge \cdots \wedge \ t_n = t_n' \ \rightarrow \ t_{n+1} = t_{n+1}' \ \vee \cdots \vee \ t_m = t_m'.$$

Therefore, the main difference between quantifier-free axioms and conditional equations is whether there are disjunctions allowed on the right hand side of implications.

[3] The results of the paper remain valid if in the definition of initial algebra semantics $Alg(SP) := \{A \in Alg(\Sigma) \mid A \models \Phi\}$ is used instead of $Mod(SP)$.

[4] Our definition of monomorphicity differs from the one given in [Wir90] where *'exactly one'* is required instead of *'at most one'*.

While these disjunctions express a kind of indeterminism, on the other hand, assuming that the quantifier-free specification is monomorphic, everything is determined by the axioms. Consequently, one might be tempted to suppose that a "deterministic" specification can always be written without those disjunctions, i.e. with conditional equations and (disjunctions of) negated equations (which correspond to implications with empty right hand side) only. The latter are covered by the initial constraint and can thus be omitted. Hence, this informal argument suggests that any data type specifiable with quantifier-free axioms and loose semantics can also be specified with conditional equations under initial semantics, i.e. just the opposite of our claim.

Another reason for our result being somewhat surprising is that there are several classical characterizations for theories to be equivalent to a set of conditional equations (e.g. closure of the models against direct products), see e.g. [CK90, Section 6.2], [Mak87, Theorem 3.5], [Fag80], and [Mal71]. Since we start from a quite restrictive condition, namely the monomorphic specifiability without quantifiers, one might hope that one of these characterizations could be applied.

Finally, many data types which at first sight seem to be promising candidates for being specifiable using loose methods but not using initial methods turn out to possess a (tricky) specification with conditional equations. For instance, Bergstra & Meyer [BM84] demonstrated this for the data type of finite sets of integers equipped with a cardinality function, although one would at first suppose that (the statement of) the axiom

$$\mathtt{insert}(\mathtt{x},\mathtt{s}) \neq \mathtt{s} \ \rightarrow\ \mathtt{card}(\mathtt{insert}(\mathtt{x},\mathtt{s})) = \mathtt{succ}(\mathtt{card}(\mathtt{s}))$$

cannot be expressed using conditional equations.[5]

4 The data type of segments

In this section we provide an example of a data type, named **SEGS**, that can be monomorphically specified using quantifier-free axioms but fails to possess a conditional equational specification under initial semantics. Roughly speaking, **SEGS** (for *seg*ment*s*) is the data type of finite initial segments of the natural numbers where the set *Seg* of segments is defined as follows:

$$\mathit{Seg} \ :=\ \Big\{\{0\},\ \{0,1\},\ \{0,1,2\},\ \{0,1,2,3\},\ \ldots\Big\}.$$

[5] The trick is to code inequalities in equational language, e.g. the inequality $\mathtt{x} \neq \mathtt{y}$ by $\mathtt{card}(\mathtt{insert}(\mathtt{x},\mathtt{insert}(\mathtt{y},\mathtt{empty}))) = \mathtt{succ}(\mathtt{succ}(\mathtt{zero}))$. Similar techniques can be applied for other data types; for further examples see e.g. [Sch98, Section 10.1].

To formally define the data type **SEGS**, we fix the signature:

$$
\boxed{\begin{array}{l}
\textbf{signature } \Sigma_{SEGS} \\
\quad \begin{array}{ll}
\textbf{sorts} & \mathtt{Nat}, \mathtt{Set}, \mathtt{Seg} \\
\textbf{functions} & \begin{array}{lll}
\mathtt{zero} & : & \to \mathtt{Nat} \\
\mathtt{succ} & : \mathtt{Nat} & \to \mathtt{Nat} \\
\mathtt{empty} & : & \to \mathtt{Set} \\
\mathtt{insert} & : \mathtt{Nat} \times \mathtt{Set} & \to \mathtt{Set} \\
\mathtt{red} & : \mathtt{Set} & \to \mathtt{Seg}
\end{array}
\end{array} \\
\textbf{end signature}
\end{array}}
$$

and declare the interpretation as follows:

$$
\begin{array}{lcl}
\mathtt{Nat}^{\mathbf{SEGS}} & := & \mathbb{N} = \{0, 1, 2, \ldots\} \\
\mathtt{Set}^{\mathbf{SEGS}} & := & \{M \subset \mathbb{N} \mid M \text{ finite}\} \\
\mathtt{Seg}^{\mathbf{SEGS}} & := & \{\emptyset\} \cup \mathit{Seg} \\
\mathtt{zero}^{\mathbf{SEGS}} & := & 0 \\
\mathtt{succ}^{\mathbf{SEGS}}(n) & := & n + 1 \\
\mathtt{empty}^{\mathbf{SEGS}} & := & \emptyset \\
\mathtt{insert}^{\mathbf{SEGS}}(n, M) & := & M \cup \{n\} \\
\mathtt{red}^{\mathbf{SEGS}}(M) & := & \begin{cases} M & \text{, if } M \in \mathit{Seg} \\ \emptyset & \text{, otherwise.} \end{cases}
\end{array}
$$

For the construction of the data type, we borrowed from two ideas:

- First, in finite model theory (cf. e.g. [EF95]), one mostly considers *finite graphs* as models because these are in some sense the most general structures. Furthermore, *connectedness* of graphs is a property of specific importance (cf. e.g. [Gai82, p. 124], [FSV95, SECTION 5], or [EF95]). Our example is similar: the segments can be regarded as connected, one-dimensional graphs.
- Second, we adopted a technique which has been applied by Thatcher, Wagner & Wright [TWW82, SECTION 3] in order to prove that conditional equations are strictly more powerful than equations. It uses "*reduction functions*" (here the function `red`) to map the problem of defining a predicate onto the problem of defining equality (which is the crucial point in algebraic specification).

Our claim is that the data type **SEGS** has a quantifier-free specification under loose semantics but no conditional specification under initial semantics. The following two sections are devoted to its proof.

5 Specifiability using loose methods

We start with the simpler task, namely the proof of specifiability, which is done by constructing a suitable specification.

Theorem 2. *There is a quantifier-free specification SP which specifies* **SEGS** *under loose semantics, i.e.* $Mod(SP) = [\mathbf{SEGS}]$.

Before providing an appropriate specification, let us agree on two abbreviations in order to improve readability:[6]

$$\begin{array}{lcl} member(\mathtt{x},\mathtt{u}) & :\equiv & \Big(\mathtt{insert}(\mathtt{x},\mathtt{u}) = \mathtt{u}\Big) \\ segp(\mathtt{u}) & :\equiv & \mathtt{red}(\mathtt{u}) \neq \mathtt{red}(\mathtt{empty}). \end{array}$$

Using these abbreviations, we can specify **SEGS** as follows:

specification SP_{SEGS}

signature Σ_{SEGS}

variables $\mathtt{x},\mathtt{y} : \mathtt{Nat}$
$\mathtt{u},\mathtt{v} : \mathtt{Set}$

axioms
(1) $\mathtt{zero} \neq \mathtt{succ}(\mathtt{x})$
(2) $\mathtt{succ}(\mathtt{x}) = \mathtt{succ}(\mathtt{y}) \rightarrow \mathtt{x} = \mathtt{y}$
(3) $\neg\, member(\mathtt{x},\mathtt{empty})$
(4) $member(\mathtt{x},\mathtt{insert}(\mathtt{y},\mathtt{u})) \leftrightarrow (\mathtt{x} = \mathtt{y} \vee member(\mathtt{x},\mathtt{u}))$
(5) $\mathtt{insert}(\mathtt{x},\mathtt{insert}(\mathtt{y},\mathtt{u})) = \mathtt{insert}(\mathtt{y},\mathtt{insert}(\mathtt{x},\mathtt{u}))$
(6) $segp(\mathtt{insert}(\mathtt{zero},\mathtt{empty}))$
(7) $segp(\mathtt{u}) \wedge member(\mathtt{x},\mathtt{u}) \rightarrow segp(\mathtt{insert}(\mathtt{succ}(\mathtt{x}),\mathtt{u}))$
(8) $member(\mathtt{succ}(\mathtt{x}),\mathtt{u}) \wedge \neg\, member(\mathtt{x},\mathtt{u}) \rightarrow \neg\, segp(\mathtt{u})$
(9) $segp(\mathtt{u}) \wedge \mathtt{u} \neq \mathtt{v} \rightarrow \mathtt{red}(\mathtt{u}) \neq \mathtt{red}(\mathtt{v})$

end specification

Lemma 3. *The quantifier-free specification* SP_{SEGS} *specifies* **SEGS** *under loose semantics, i.e.* $Mod(SP_{SEGS}) = \mathbf{SEGS}$.

Proof. We can easily verify that all axioms are valid in **SEGS**. Therefore, **SEGS** is a model of SP_{SEGS} and it remains to prove the monomorphicity of SP_{SEGS}.

For that purpose, we use the fact that a specification *SP* is monomorphic iff *SP* fixes the ground equations, i.e. iff for all ground equations $t_1 = t_2$, either $SP \models t_1 = t_2$ or $SP \models t_1 \neq t_2$ (cf. [Wir90, FACT 2.3.2]). We check separately every kind of ground equation:

(a) ground equations of the sort `Nat`:
Due to axioms (1) and (2), we get $SP \models t_1 \neq t_2$ whenever t_1 and t_2 differ syntactically; otherwise, $SP \models t_1 = t_2$ holds trivially.

[6] Note, that we are not adding any new symbols to the signature.

(b) ground equations of the sort `Set`:
We first consider equations of the form $member(t,t')$. These are fixed due to the inductive definition of *member* in axioms (3) and (4). Now, let $t_1 = t_2$ be an arbitrary ground equation of the sort `Set`. In the case that there is a ground term t with $SP_{SEGS} \models member(t,t_1)$ but $SP_{SEGS} \models \neg\, member(t,t_2)$ (or vice versa) we immediately obtain $SP_{SEGS} \models t_1 \neq t_2$. Otherwise,

$$SP_{SEGS} \models member(t,t_1) \iff SP_{SEGS} \models member(t,t_2)$$

holds for all ground terms t of the sort `Nat`. A proof by induction using axiom (5) yields $SP_{SEGS} \models t_1 = t_2$.

(c) ground equations of the sort `Seg`:
We start with ground equations of the form $segp(t)$. If $t^{\mathbf{SEGS}} \in Seg$, we obtain $SP_{SEGS} \models segp(t)$ using axioms (6) and (7). Otherwise, axiom (8) implies $SP_{SEGS} \models \neg\, segp(t)$.
Now let $\mathtt{red}(t_1) = \mathtt{red}(t_2)$ be an arbitrary ground equation. In case that $SP_{SEGS} \models t_1 = t_2$, we immediately obtain $SP_{SEGS} \models \mathtt{red}(t_1) = \mathtt{red}(t_2)$. The same holds in case that $SP_{SEGS} \models \neg\, segp(t_1)$ and $SP_{SEGS} \models \neg\, segp(t_2)$. In all remaining cases, we get $SP_{SEGS} \models \mathtt{red}(t_1) \neq \mathtt{red}(t_2)$ by applying axiom (9).

■

6 Non-Specifiability using initial methods

Having proved that the data type **SEGS** can be monomorphically specified without quantifiers, in order to establish theorem 1, it remains to show that **SEGS** fails to possess a conditional equational specification under initial semantics. This is formalized in the following theorem.

Theorem 4. *There is no conditional equational specification SP which specifies* **SEGS** *under initial semantics, i.e.* $I(SP) = [\mathbf{SEGS}]$.

The idea for the proof is to construct non-desired models for possible conditional equational axiomatizations. As non-desired models, we consider the data types $\mathbf{SEGS}_m$ which differ from **SEGS** in that they also treat some non-segments like segments, namely those sets $M \subseteq \mathbb{N}$ which have exactly one gap of length one sufficiently far away from 0. More precisely, we define $\mathbf{SEGS}_m$ just as **SEGS**, except for:

$$\begin{aligned}
\mathtt{Seg}^{\mathbf{SEGS}_m} &:= Seg \cup \Big\{ M \setminus \{n\} \;\Big|\; M \in Seg,\ n \geq m \Big\} \\
\mathtt{red}^{\mathbf{SEGS}_m}(M) &:= \begin{cases} M & \text{, if } M \in Seg \\ M & \text{, if } M \cup \{n\} \in Seg \text{ for some } n \geq m \\ \emptyset & \text{, otherwise.} \end{cases}
\end{aligned}$$

For any conditional equation φ, we determine a number m such that

$$\mathbf{SEGS} \models \varphi \implies \mathbf{SEGS}_m \models \varphi. \qquad (\star)$$

In order to choose m, it suffices to consider the number $\mathit{inserts}(\varphi)$ of occurrences of the symbol `insert` in φ. We are going to prove that $(\star)$ holds whenever $m > \mathit{inserts}(\varphi)$. This is done by constructing modifications v' of valuations v such that $\mathbf{SEGS}_m, v \models \neg\varphi$ implies $\mathbf{SEGS}, v' \models \neg\varphi$.

The proof of theorem 4 needs some preparations. First, we show that we can restrict ourselves to conditional equations of a certain "simple" form. More precisely, we call a conditional Σ_{SEGS}-equation

$$t_1 = t_1' \ \wedge \cdots \wedge \ t_k = t_k' \ \rightarrow \ t = t'$$

simple if it has the following properties:

- if $t_i = t_i'$ (with $i \in \{1, \ldots, k\}$) is a `Seg`-equation, then it is of the form $\mathtt{red}(x) = \mathtt{red}(\mathtt{empty})$ for some variable x of the sort `Set`, and
- if $t = t'$ is a `Seg`-equation, then it is of the form $\mathtt{red}(x) = \mathtt{red}(y)$ for some variables x, y of the sort the `Set`.

Lemma 5. *For a finite set Φ of conditional equations over Σ_{SEGS}, there is a finite set Φ' of* simple *conditional equations over Σ_{SEGS} such that:*[7]

- $A \models \Phi' \iff A \models \Phi$ *for all* $A \in \{\mathbf{SEGS}\} \cup \{\mathbf{SEGS}_m \mid m \in \mathbb{N}\}$
- $\mathit{inserts}(\Phi') = \mathit{inserts}(\Phi)$.

Proof. First, we eliminate variables of the sort `Seg`: if y is such a variable, we can replace all occurrences of y by the term $\mathtt{red}(x)$, where x is a new variable of the sort `Set`. This does not affect validity in *term-generated* models.

Therefore, we can assume `Seg`-equations to be of the form $\mathtt{red}(t_1) = \mathtt{red}(t_2)$. Moreover, we can assume the terms t_1, t_2 of the sort `Set` to be variables, because the following transformations of conditional equations preserve equivalence:

$$\begin{array}{l}(\cdots \wedge \mathtt{red}(t_1) = \mathtt{red}(t_2) \wedge \cdots \rightarrow \cdots) \\ \rightsquigarrow \quad (\cdots \wedge x_1 = t_1 \wedge x_2 = t_2 \wedge \mathtt{red}(x_1) = \mathtt{red}(x_2) \wedge \cdots \rightarrow \cdots)\end{array}$$

$$\begin{array}{l}(\cdots \rightarrow \mathtt{red}(t_1) = \mathtt{red}(t_2)) \\ \rightsquigarrow \quad (\cdots \wedge x_1 = t_1 \wedge x_2 = t_2 \rightarrow \mathtt{red}(x_1) = \mathtt{red}(x_2)).\end{array}$$

(x_1, x_2 are new variables of the sort `Set`.)

It remains to prove that the `Seg`-equations on the left hand side of implications can even be transformed to the form $\mathtt{red}(x) = \mathtt{red}(\mathtt{empty})$. To show this, we make use of the fact that for all $A \in \{\mathbf{SEGS}\} \cup \{\mathbf{SEGS}_m \mid m \in \mathbb{N}\}$ and all valuations v:

$$\begin{array}{l}A, v \models \mathtt{red}(x_1) = \mathtt{red}(x_2) \\ \quad \iff \quad A, v \models x_1 = x_2 \vee (\mathtt{red}(x_1) = \mathtt{red}(\mathtt{empty}) \wedge \mathtt{red}(x_2) = \mathtt{red}(\mathtt{empty})).\end{array}$$

Hence, a conditional equation of the form

$$\cdots \wedge \mathtt{red}(x_1) = \mathtt{red}(x_2) \wedge \cdots \rightarrow \cdots$$

[7] $\mathit{inserts}(\Phi) := \max\{\mathit{inserts}(\varphi) \mid \varphi \in \Phi\}$.

can be replaced by the following two conditional equations:

$$\cdots \wedge x_1 = x_2 \wedge \cdots \to \cdots$$
$$\cdots \wedge \mathtt{red}(x_1) = \mathtt{red}(\mathtt{empty}) \wedge \mathtt{red}(x_2) = \mathtt{red}(\mathtt{empty}) \wedge \cdots \to \cdots$$

It is easily seen that none of the applied transformations affects the number of `insert` symbols. ■

Most easy cases in the final proof can be dealt with using the following fact.

Lemma 6. *Let* $v : X \to \mathbf{SEGS}$ *be a valuation and* $m \in \mathbb{N}$. *Then*

(a) *for any equation* $t_1 = t_2$ *of the sort* `Nat` *or* `Set`*:*

$$\mathbf{SEGS}_m, v \models t_1 = t_2 \iff \mathbf{SEGS}, v \models t_1 = t_2.$$

(b) *for variables* x *of the sort* `Set`*:*

$$\mathbf{SEGS}_m, v \models \mathtt{red}(x) = \mathtt{red}(\mathtt{empty}) \implies \mathbf{SEGS}, v \models \mathtt{red}(x) = \mathtt{red}(\mathtt{empty}).$$

Proof. Statement (a) is obvious, since $\mathbf{SEGS}_m$ and $\mathbf{SEGS}$ do not differ in their evaluation of terms of the sort `Nat` or `Set`.

To prove (b) we simply observe that $\mathbf{SEGS}_m, v \models \mathtt{red}(x) = \mathtt{red}(\mathtt{empty})$ implies $v(x) \notin \mathit{Seg}$ and thus $\mathbf{SEGS}, v \models \mathtt{red}(x) = \mathtt{red}(\mathtt{empty})$. ■

Since the modification of the valuation should preserve `Set`-equalities, we have to change `Set`-variables simultaneously if they are "connected" through equations. Let us define this relationship formally. Two variables x_1, x_2 of the sort `Set` are said to be *adjacent* in a set Φ of `Set`-equations if there is an equation of the form $t_1(x_1) = t_2(x_2)$ in Φ. Adjacency between `Set`-variables and the symbol `empty` is defined similarly.

The reflexive, symmetric, transitive closure of adjacency (in Φ) is obviously an equivalence relation on `Set`-variables and the symbol `empty`. We denote it by $\sim_\Phi$. The equivalence class $[\mathtt{empty}]_{\sim_\Phi} = \{x \mid x \sim_\Phi \mathtt{empty}\}$ will be of particular interest, allowing to formulate the following estimation.

Lemma 7. *Let* Φ *be a set of equations of the sort* `Set` *and* v *a valuation such that* $\mathbf{SEGS}_m, v \models \Phi$. *Then,* $|v(x)| \leq \mathit{inserts}(\Phi)$ *for all* `Set`*-variables* $x \in [\mathtt{empty}]_{\sim_\Phi}$.

Proof. Let $x \in [\mathtt{empty}]_{\sim_\Phi}$. Then there is a sequence $(x_1, x_2, \ldots, x_k)$ of distinct `Set`-variables such that $x_k :\equiv x$ and Φ contains distinct equations of the form:

$$t_1(\mathtt{empty}) = t'_1(x_1), \quad t_2(x_1) = t'_2(x_2), \quad \ldots, \quad t_k(x_{k-1}) = t'_k(x_k).$$

For any $i \in \{1, \ldots, k\}$ (with $x_0 :\equiv \mathtt{empty}$ and $v(\mathtt{empty}) := \emptyset$), one obtains $|v(x_i)| \leq \mathit{inserts}(t_i) + |v(x_{i-1})|$ as well as $|v(x_{i-1})| \leq \mathit{inserts}(t'_i) + |v(x_i)|$, and therefore

$$\Big| |v(x_i)| - |v(x_{i-1})| \Big| \;\leq\; \mathit{inserts}\Big(t_i(x_{i-1}) = t'_i(x_i)\Big).$$

Using the above inequation, the desired estimation is derived as follows:

$$\begin{aligned}
|v(x)| &= |v(x_k)| - |v(x_0)| \\
&= \sum_{i\in\{1,\ldots,k\}} |v(x_i)| - |v(x_{i-1})| \\
&\leq \sum_{i\in\{1,\ldots,k\}} \Big| |v(x_i)| - |v(x_{i-1})| \Big| \\
&\leq \sum_{i\in\{1,\ldots,k\}} \mathit{inserts}\Big(t_i(x_{i-1}) = t'_i(x_i)\Big) \\
&\leq \mathit{inserts}(\Phi).
\end{aligned}$$

■

These preparations allow us to prove the following key lemma.

Lemma 8. *Let φ be a* simple *conditional equation over the signature Σ_{SEGS} and $m > \mathit{inserts}(\varphi)$. Then*

$$\mathbf{SEGS} \models \varphi \quad \Longrightarrow \quad \mathbf{SEGS}_m \models \varphi.$$

Proof. Let φ be a *simple* conditional equation, i.e. of the form:

$$\varphi \;\equiv\; \Big(t_1 = t'_1 \;\wedge \cdots \wedge\; t_k = t'_k \;\rightarrow\; t = t'\Big).$$

We will show the implication

$$\mathbf{SEGS} \models \varphi \quad \Longrightarrow \quad \mathbf{SEGS}_m \models \varphi$$

for $m > \mathit{inserts}(\varphi)$ by contraposition and therefore assume $\mathbf{SEGS}_m \not\models \varphi$. Hence, there is a valuation $v : X \to \mathbf{SEGS}_m$ such that $\mathbf{SEGS}_m, v \models \neg\varphi$, i.e. $\mathbf{SEGS}_m, v \models t_1 = t'_1 \wedge \cdots \wedge t_k = t'_k$ and $\mathbf{SEGS}_m, v \models t \neq t'$.

If $t = t'$ is an equation of the sort `Nat` or `Set`, we use lemma 6(a) to obtain $\mathbf{SEGS}, v \models t \neq t'$, and lemma 6(a) and (b) for $\mathbf{SEGS}, v \models t_1 = t'_1 \wedge \cdots \wedge t_k = t'_k$. Therefore, $\mathbf{SEGS}, v \models \neg\varphi$, i.e. $\mathbf{SEGS} \not\models \varphi$.

If $\mathbf{SEGS}, v \models t \neq t'$, we can again use lemma 6 to obtain $\mathbf{SEGS}, v \models \neg\varphi$, i.e. $\mathbf{SEGS} \not\models \varphi$. Therefore, we can focus on the (interesting) case that $t = t'$ is a `Seg`-equation with $\mathbf{SEGS}, v \models t = t'$. Since φ is simple, the equation $t = t'$ has to be of the form $\mathtt{red}(x_1) = \mathtt{red}(x_2)$ for appropriate `Set`-variables x_1, x_2. Together with $\mathbf{SEGS}_m, v \models \mathtt{red}(x_1) \neq \mathtt{red}(x_2)$ and $\mathbf{SEGS}, v \models \mathtt{red}(x_1) = \mathtt{red}(x_2)$, the definitions of **SEGS** and $\mathbf{SEGS}_m$ yield:

(a) $v(x_1) \neq v(x_2)$,
because otherwise, $\mathbf{SEGS}_m, v \models \mathtt{red}(x_1) = \mathtt{red}(x_2)$.

(b) $v(x_1), v(x_2) \notin \mathit{Seg}$,
otherwise, the set $v(x_1)$ or the set $v(x_2)$ would be a segment, and (a) would imply $\mathbf{SEGS}, v \models \mathtt{red}(x_1) \neq \mathtt{red}(x_2)$.

(c) there is some $n \geq m$ such that $v(x_1) \cup \{n\} \in \mathit{Seg}$ or $v(x_2) \cup \{n\} \in \mathit{Seg}$; w.l.o.g. (if necessary, exchange x_1 and x_2) let $v(x_1) \cup \{n\} \in \mathit{Seg}$, because otherwise, (b) would yield $\mathbf{SEGS}_m, v \models \mathtt{red}(x_1) = \mathtt{red}(\mathtt{empty})$ as well as $\mathbf{SEGS}_m, v \models \mathtt{red}(x_2) = \mathtt{red}(\mathtt{empty})$. Obviously this implies $\mathbf{SEGS}_m, v \models \mathtt{red}(x_1) = \mathtt{red}(x_2)$.

(d) $|v(x_1)| > m$, because, due to statements (b) and (c), the set $v(x_1)$ has to be of the form $\{0, 1, \ldots, n-1, n+1, \ldots, l\}$ for some $l \geq n+1 \geq m+1$.

Now, let Φ denote the set of Set-equations in $t_1 = t'_1 \wedge \cdots \wedge t_k = t'_k$. We define a modified valuation v' as follows:

$$v'(x) := \begin{cases} v(x) \cup \{n\} & \text{, if } x \text{ is a variable of sort } \mathtt{Set} \text{ such that } x \notin [\mathtt{empty}]_{\sim_\Phi} \\ v(x) & \text{, otherwise} \end{cases}$$

This modified valuation has the following properties:

(e) $v'(x_1) \in \mathit{Seg}$, because $x_1 \notin [\mathtt{empty}]_{\sim_\Phi}$ by (d) and lemma 7, and therefore, by (c), even $v'(x_1) = v(x_1) \cup \{n\} \in \mathit{Seg}$.

(f) $v'(x_1) \neq v'(x_2)$, otherwise, $v(x_1) \cup \{n\} = v(x_2) \cup \{n\}$ (even if $x_2 \in [\mathtt{empty}]_{\sim_\Phi}$ because then, $v'(x_2) = v(x_2) = v(x_1) \cup \{n\}$ would imply $n \in v(x_2)$). Hence, we obtain $v(x_2) = v(x_1) \cup \{n\}$ or $v(x_2) = v(x_1)$, contradicting (b) or (a), respectively.

(g) $\mathbf{SEGS}, v' \models \mathtt{red}(x_1) \neq \mathtt{red}(x_2)$, which follows from (e) and (f) and the definition of $\mathtt{red}^{\mathbf{SEGS}}$.

It remains to prove that

(h) $\mathbf{SEGS}, v' \models t_1 = t'_1 \wedge \cdots \wedge t_k = t'_k$.
We consider each of the equations $t_i = t'_i$ separately and distinguish several cases using the precondition $\mathbf{SEGS}_m, v \models t_i = t'_i$ in each of them:

- If $t_i = t'_i$ is a Nat-equation, lemma 6(a) yields $\mathbf{SEGS}, v \models t_i = t'_i$. Hence, we obtain $\mathbf{SEGS}, v' \models t_i = t'_i$, because the evaluations of Nat-variables under v and v' are identical.
- If $t_i = t'_i$ is an equation of the sort Set, we again distinguish two cases. If $t_i = t'_i$ contains the symbol empty or a variable from $[\mathtt{empty}]_{\sim_\Phi}$, then all Set-variables occurring in $t_i = t'_i$ belong to $[\mathtt{empty}]_{\sim_\Phi}$ (due to the definition of $\sim_\Phi$). As v' coincides with v on these variables, we can use the same argument as for Nat-equations.
If $t_i = t'_i$ contains neither the symbol empty nor a Set-variable from $[\mathtt{empty}]_{\sim_\Phi}$, then $t_i = t'_i$ is of the form $t_i(x_i) = t'_i(x'_i)$ for appropriate variables x_i, x'_i with $x_i, x'_i \notin [\mathtt{empty}]_{\sim_\Phi}$. Thus, $v'(x_i) = v(x_i) \cup \{n\}$ and also $v'(x'_i) = v(x'_i) \cup \{n\}$. Together with $\mathbf{SEGS}, v \models t_i(x_i) = t'_i(x'_i)$, we get $\mathbf{SEGS}, v' \models t_i(x_i) = t'_i(x'_i)$.

- If $t_i = t'_i$ is an equation of the sort Seg, since φ is simple, it must be of the form $\mathtt{red}(x) = \mathtt{red}(\mathtt{empty})$.
  In case that $x \in [\mathtt{empty}]_{\sim_\Phi}$, one obtains $v'(x) = v(x)$, and it follows from the precondition that $\mathbf{SEGS}_m, v' \models \mathtt{red}(x) = \mathtt{red}(\mathtt{empty})$. Applying lemma 6(b) shows $\mathbf{SEGS}, v' \models \mathtt{red}(x) = \mathtt{red}(\mathtt{empty})$.
  In case that $x \notin [\mathtt{empty}]_{\sim_\Phi}$, the definition of v' yields $v'(x) = v(x) \cup \{n\}$. For $v(x) = \emptyset$, we obtain $v'(x) \notin \mathit{Seg}$ because of $n \geq m > 0$. Otherwise, if $v(x) \neq \emptyset$, the precondition $\mathbf{SEGS}_m, v \models \mathtt{red}(x) = \mathtt{red}(\mathtt{empty})$ implies $v'(x) \notin \mathit{Seg}$. Thus, $v'(x) \notin \mathit{Seg}$ holds in both cases, and therefore $\mathbf{SEGS}, v' \models \mathtt{red}(x) = \mathtt{red}(\mathtt{empty})$.

Statements (g) and (h) imply $\mathbf{SEGS}, v' \models \neg\varphi$, and thus $\mathbf{SEGS} \not\models \varphi$. ■

Now, we can turn to the proof of the main theorem.

Proof of theorem 4. For contradiction, we assume that there is a conditional equational specification $SP = (\Sigma_{SEGS}, \Phi)$ with $I(SP) = [\mathbf{SEGS}]$. Moreover, let Φ' denote the set of *simple* conditional equations constructed from Φ as in lemma 5, and $m > \mathit{inserts}(\Phi) = \mathit{inserts}(\Phi')$.

By lemma 8, it follows that $\mathbf{SEGS}_m \models \Phi'$, and thus $\mathbf{SEGS}_m \models \Phi$ (see lemma 5).

On the other hand, it can be easily verified that there is no homomorphism $h : \mathbf{SEGS} \to \mathbf{SEGS}_m$. For $M_m := \{0, 1, \ldots, m-1, m+1\}$ one would obtain

$$\begin{aligned} M_m &= \mathtt{red}^{\mathbf{SEGS}_m}(M_m) \\ &= \mathtt{red}^{\mathbf{SEGS}_m}(h(M_m)) \\ &= h(\mathtt{red}^{\mathbf{SEGS}}(M_m)) \\ &= h(\emptyset) \\ &= \emptyset. \end{aligned}$$

Consequently, the data type **SEGS** is not initial in $\mathit{Mod}(SP)$—contradicting the assumption. ■

7 Conclusions

We have provided an example of a computable data type which possesses a quantifier-free monomorphic specification, but fails to have a conditional equational specification under initial semantics (both without hidden functions). Our result is in the line of the work done by Majster [Maj79], Bergstra & Tucker [BT87], Thatcher, Wagner & Wright [TWW82] and others (e.g. [Ore79, BM84, Ber93]) and answers an important question which has been left open. Roughly speaking, it shows that the restriction to conditional equations—which is standard in the initial approach to algebraic specification—means a restriction in expressiveness.

However, the result is of relevance not only in the scope of algebraic specification. For instance, it can be read as a result in proof theory. This is due to the fact that both the model of a monomorphic quantifier-free specification and the initial model of conditional equations can be characterized by the ground equations derivable from the axioms. Thus, in proof-theoretic terms, the question we dealt with is: Given a finite, quantifier-free set Φ of formulas (over a functional signature which allows to construct a ground term for any sort) such that for any two ground terms t_1, t_2 of the same sort either $t_1 = t_2$ or $t_1 \neq t_2$ is derivable. Is there a finite set Φ' of conditional equations (over the same signature) such that exactly the same ground equations are derivable from Φ'? Theorem 1 answers this question negatively.

Furthermore, our result might be of some interest in the context of logic programming and rewriting: it provides further justification for the generalizations to *disjunctive* logic programming (e.g. [LMR92, TZ92, NL95]) and to *quasi*-Horn clause rewriting (e.g. [Kap88, BG94]).

References

[Ber93] R. Berghammer. On the characterization of the integers: the hidden function problem revisited. *Acta Cybernetica*, 11(1-2):85–96, 1993.

[BG94] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[BM84] J.A. Bergstra and J.J. Meyer. On specifying sets of integers. *Elektron. Informationsverarb. u. Kybernet.*, 20:532–541, 1984.

[BT82] J.A. Bergstra and J.V. Tucker. The completeness of the algebraic specification methods for computable data types. *Information and Control*, 54:186–200, 1982.

[BT87] J.A. Bergstra and J.V. Tucker. Algebraic specifications of computable and semicomputable data types. *Theoret. Comput. Sci.*, 50:137–181, 1987.

[CK90] C.C. Chang and H.J. Keisler. *Model Theory*, volume 73 of *Studies in Logic and the Found. of Math.* Elsevier Science Publishers B. V., 1990.

[EF95] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Perspectives in Mathematical Logic. Springer, Berlin, 1995.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1, Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science.* Springer, Berlin, 1985.

[Fag80] R. Fagin. Horn clauses and database dependencies (extended abstract). In *Conference Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, pages 123–134, 1980.

[FSV95] R. Fagin, L. Stockmeyer, and M. Vardi. On monadic NP vs. monadic co-NP. *Information and Computation*, 120:78–92, 1995.

[Gai82] H. Gaifman. On local and non-local properties. In J. Stern, editor, *Proc. of the Herbrand Symposium, Logic Colloquium'81*, pages 105–135. North-Holland, 1982.

[GGM76] V. Giarratana, F. Gimona, and U. Montanari. Observability concepts in abstract data type specifications. In *5th Symposium on Mathematical Foundations of Computer Science*, volume 45 of *LNCS*, pages 576–587. Springer, Berlin, 1976.

[GTWW75] J.A. Goguen, J.W. Thatcher, E.G. Wagner, and J.B. Wright. Abstract data types as initial algebras and the correctness of data representations. In *Proc. Conf. on Computer Graphics, Pattern Recognition and Data Structures*, pages 89–93, 1975.

[Kap88] S. Kaplan. Positive/negative conditional rewriting. In M.P. Chytil, L. Janiga, and V. Koubek, editors, *Mathematical Foundations of Computer Science*, volume 324 of *LNCS*, pages 381–395. Springer, Berlin, 1988.

[KS98] D. Kempe and A. Schönegge. On the power of quantifiers in first-order algebraic specification. In *Annual Conference of the European Association for Computer Science Logic*, 1998.

[LEW96] J. Loeckx, H.-D. Ehrich, and M. Wolf. *Specification of abstract data types.* Verlag Wiley-Teubner, 1996.

[LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT-Press, 1992.

[Maj79] M.E. Majster. Data types, abstract data types and their specification problem. *Theoret. Comput. Sci.*, 8:89–127, 1979.

[Mak87] J.A. Makowsky. Why Horn formulas matter in computer science: Initial structures and generic examples. *J. Comput. Sys. Sci.*, 34:266–292, 1987.

[Mal71] A.I. Malćev. *The Metamathematics of Algebraic Systems: Collected Papers 1936-1967.* Translated and edited by B.F. Wells III. North-Holland, Amsterdam, 1971.

[NL95] G. Nadathur and D.W. Loveland. Uniform proofs and disjunctive logic programming. In D. Kozen, editor, *Proc. of the Tenth Annual Symp. on Logic in Comp. Sci.*, pages 148–155. IEEE Computer Society Press, 1995.

[Ore79] F. Orejas. On the power of conditional specifications. *ACM SIGPLAN Notices*, 14(7):78–81, 1979.

[Sch97] A. Schönegge. The hidden function question revisited. In *Proceedings of the 6th International Conference on Algebraic Methodology and Software Technology*, volume 1349 of *LNCS*, pages 451–464. Springer, Berlin, 1997.

[Sch98] A. Schönegge. *Specifiability of computable data types (in german).* PhD thesis, Universität Karlsruhe, Fakultät für Informatik, Shaker Verlag, 1998.

[SW98] D.T. Sannella and M. Wirsing. Specification languages. In E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, editors, *Algebraic Foundations of Systems Specifications.* Chapman and Hall, 1998. To appear.

[TWW82] J.W. Thatcher, E.G. Wagner, and J.B. Wright. Data type specification: Parameterization and the power of specification techniques. In *ACM Trans. on Prog. Languages and Systems*, volume 4, pages 711–732, 1982.

[TZ92] J.V. Tucker and J.I. Zucker. Deterministic and nondeterministic computation and Horn programs, on abstract data types. *Journal of Logic Programming*, 13(1):23–55, 1992.

[Wan81] M. Wand. Final algebra semantics and data type extensions. *J. Comput. System Sci.*, 19:27–44, 1981.

[Wir90] M. Wirsing. *Algebraic Specification*, volume B of *Handbook of Theoretical Computer Science*, chapter 13. Elsevier Science Publishers B. V., 1990.

[Wir95] M. Wirsing. Algebraic specification languages: an overview. In *Proc. 10th Workshop on Specification of Abstract Data Types*, volume 906 of *LNCS*, pages 81–115. Springer, Berlin, 1995.

[WPP+83] M. Wirsing, P. Pepper, H. Partsch, W. Dosch, and M. Broy. On hierarchies of abstract data types. *Acta Informatica*, 20:1–33, 1983.