On the power of quantifiers in first-order algebraic specification*

David Kempe and Arno Schönegge

Institut für Logik, Komplexität und Deduktionssysteme
Universität Karlsruhe, 76128 Karlsruhe, Germany
{kempe|schoenegge}@ira.uka.de

Abstract. The well-known completeness theorem of Bergstra & Tucker [BT82, BT87] states that all computable data types can be specified without quantifiers, i.e., quantifiers can be dispensed with—at least if the introduction of auxiliary (hidden) functions is allowed.
However, the situation concerning the specification *without* hidden functions is quite different. Our main result is that, in this case, quantifiers *do* contribute to expressiveness. More precisely, we give an example of a computable data type that has a monomorphic first-order specification (without hidden functions) and prove that it fails to possess a monomorphic quantifier-free specification (without hidden functions).

1 Introduction

The expressive power of first-order algebraic specification methods has been extensively studied over several years, e.g., by Majster [Maj79], Bergstra & Tucker [BT82, BT87], the ADJ-group [TWW82], and others [Ore79, BBTW81, Wir90].

One main result is that, if equipped with hiding mechanisms, all common algebraic specification methods are adequate for computable[1] data types. 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. Bergstra & Tucker [BT82, BT87] proved that any computable algebra A possesses a quantifier-free specification involving at most $2n+3$ hidden functions (where n is the number of sorts in A) that defines A under both its initial and final algebra semantics. Hence, by using hiding mechanisms, a monomorphic quantifier-free specification can be constructed for any computable data type.[2]

Consequently, if one is interested in specifying computable data types only (which is reasonable in most practical applications), the use of quantifiers does

* This research has partly been supported by the "Deutsche Forschungsgemeinschaft" within the "Schwerpunktprogramm Deduktion". The results were obtained in the course of [Kem98].

[1] We define the notions of (computable) data types, specifiability, etc. in section 2.

[2] On the other hand, any data type defined by a monomorphic quantifier-free specification is computable (cf. [Wir90, Theorem 4.2.1]). Hence, the computability of a data type is characterized by specifiability without quantifiers.

not enhance expressiveness.[3]

However, the situation changes substantially if we do not allow hidden functions. In particular, the completeness for computable data types is lost: [Ber93, Sch97a, Sch97b] provided examples of data types that are computable but fail to possess monomorphic first-order specifications without hidden functions. The loss of expressiveness caused by disabling hiding therefore cannot be compensated completely, even if arbitrary first-order quantification is used. Naturally, the question arises whether quantification makes up for this loss at least to some extent. In other words: if a quantifier-free specification without hidden functions does not exist, can we hope to find at least a first-order specification (using quantifiers) without hidden functions?

This paper gives a positive answer by proving

Theorem 1. *There is a computable data type that possesses a monomorphic first-order specification without hidden functions but fails to have a monomorphic quantifier-free specification without hidden functions.*

While this theorem states what one would expect, it is difficult to identify examples of the required class of data types. Especially, in order to prove that a data type cannot be specified without quantifiers, non-standard models have to be constructed for any potential quantifier-free specification.

The paper is organized as follows: Section 2 recalls some basic definitions. In section 3, we demonstrate that following a naive approach does not result in examples of data types that point out the necessity of quantifiers. Section 4 presents a solution, i.e. a data type A that is claimed to have a monomorphic specification with quantifiers but none without (both not using hidden functions). The first property is established in section 5, the second in section 6. Finally, in the last section, we draw conclusions and discuss related work.

2 Preliminaries

We assume the reader to be familiar with the very basic notions of algebraic specification (cf. e.g. [Wir90] or [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 there is a denotation, i.e. a ground term $t \in T(\Sigma, \emptyset)$ with $t^A = a$, for each of its carrier elements $a \in s^A$ ($s \in S$). 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

[3] Of course, there are *non-computable* data types which can be specified using quantifiers but — as argued in footnote 2 — not without. More precisely, even all *arithmetical* data types possess a monomorphic first-order specification (cf. the proof of [Wir90, Lemma 5.3.8]).

"abstract" *data type*, is denoted by $[A]$. Sometimes, we simply speak of the *data type* A.

A Σ-algebra A 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.

The usual *satisfaction* of a Σ-formula φ by a Σ-algebra A w.r.t. a valuation $v : X \rightarrow A$ 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*. If the formulas in Φ do not contain quantifiers, we speak of *quantifier-free specification*. We adopt loose semantics and define $Mod(SP) := \{A \in Gen(\Sigma) \mid A \models \Phi\}$ to be the set of *models* of SP. If $A \models \varphi$ for all $A \in Mod(SP)$, we simply write $SP \models \varphi$.

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 $Mod(SP)$. A specification $SP = (\Sigma, \Phi)$ is said to *specify* a Σ-algebra A (and also the corresponding abstract data type $[A]$) if $Mod(SP) = [A]$. Hence, a specification SP specifies a Σ-algebra A if and only if SP is monomorphic and $A \in Mod(SP)$.

A Σ-algebra A with $\Sigma = (S, F)$ can be specified *with hidden functions* 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.

3 A first approach: primality of numbers

When looking for a suitable example to prove theorem 1, one that might come to mind immediately is that of the primality predicate: its customary definition in first-order logic looks like the following:

$$\mathrm{prime}(x) \quad \leftrightarrow \quad (x > 1) \wedge \neg \exists y, z.(y > 1 \wedge z > 1 \wedge x = yz)$$

and thus involves a quantifier. Although one might suspect that this quantifier is indispensable, it turns out that a quantifier-free specification of the primality predicate without further auxiliary functions actually exists. This result follows from a more general fact, stating the quantifier-free specifiability of every decidable predicate using the signature

signature				
sorts	`Nat, Bool`			
functions	`true, false`	:		$\to$ `Bool`
	`zero`	:		$\to$ `Nat`
	`succ`	:	`Nat`	$\to$ `Nat`
	`add, mult`	:	`Nat` $\times$ `Nat`	$\to$ `Nat`
	`char`	:	`Nat`	$\to$ `Bool`
end signature				

For every $M \subseteq \mathbb{N}$, A_M denotes the algebra that interprets `Nat`, `Bool` and their basic operations in the usual way and furthermore defines

$$\mathtt{char}^{A_M}(x) \quad = \quad \begin{cases} \mathit{true} & \text{if } x \in M \\ \mathit{false} & \text{otherwise} \end{cases}$$

to be the characteristic function of M.

Fact 2. *Let M be a* decidable *subset of the natural numbers.*[4] *Then, there exists a quantifier-free specification SP without hidden functions such that*

$$\mathit{Mod}(SP) \quad = \quad [A_M].$$

Proof. The proof is a consequence of the characterization of enumerable sets by diophantine equations. A good introduction to the subject is [Mat93].

Seeing that the basic operations on `Nat` and `Bool` can be easily specified without quantifiers, we focus on the specifiability of the function $\mathtt{char}^{A_M}$.

Since M is decidable, both M and its complement are recursively enumerable. Matiyasevich's Theorem states that all recursively enumerable sets are diophantine. Hence, M and $\mathbb{N} \setminus M$ are diophantine, meaning that there are polynomials p_1, q_1, p_2, q_2 with positive integer coefficients such that

$$\begin{array}{rcl} x \in M & \Longleftrightarrow & \text{there exist } y_1, \ldots, y_m \text{ with } p_1(x, y_1, \ldots, y_m) = q_1(x, y_1, \ldots, y_m), \\ x \notin M & \Longleftrightarrow & \text{there exist } y_1, \ldots, y_n \text{ with } p_2(x, y_1, \ldots, y_n) = q_2(x, y_1, \ldots, y_n). \end{array}$$

Since the polynomials can be written as terms using `zero`, `succ`, `add`, and `mult` only, we obtain two quantifier-free axioms specifying $\mathtt{char}^{A_M}$:

$$\begin{array}{rcl} p_1(\mathtt{x}, \mathtt{y}_1, \ldots, \mathtt{y}_m) & = & q_1(\mathtt{x}, \mathtt{y}_1, \ldots, \mathtt{y}_m) \;\to\; \mathtt{char}(\mathtt{x}) = \mathtt{true} \\ p_2(\mathtt{x}, \mathtt{y}_1, \ldots, \mathtt{y}_n) & = & q_2(\mathtt{x}, \mathtt{y}_1, \ldots, \mathtt{y}_n) \;\to\; \mathtt{char}(\mathtt{x}) = \mathtt{false}. \end{array}$$

Note that we are not introducing any new signature symbols: p_1, q_1, p_2, q_2 are merely denotations for the terms representing the polynomials. ■

Choosing M to be the set of all prime numbers, the above fact 2 yields the desired specification. In this special case, polynomials p_2, q_2 can easily be found, whereas determining p_1, q_1 requires a great amount of number theory. In [Mat93, p. 55], a polynomial representation of the set of prime numbers is given.

[4] The statement can easily be generalized to decidable subsets of $\mathbb{N}^k$ for any k and also to computable functions from $\mathbb{N}^k$ to $\mathbb{N}$.

4 The example: completeness of graphs

Looking once more at the reason why the previous section's example did indeed have a quantifier-free specification, we realize that the powerful arithmetical operations allowed us to code all decidable properties of natural numbers. Therefore, we turn to properties of structures possessing fewer operations, namely the completeness of graphs (for the same reason, graphs and their properties are often considered in finite model theory, cf. section 7).

More specifically, we consider directed graphs with a finite number of edges, and whose vertices are exactly the natural numbers. We can therefore identify a graph with the set of its edges, i.e., a *graph* G is a *finite* subset $G \subset \mathbb{N} \times \mathbb{N}$. It is called *complete* if $(n, m) \in G$ for every $n, m \in G$. Here, $n \in G$ means that there exists an $n' \in \mathbb{N}$ such that $(n, n') \in G$ or $(n', n) \in G$.

To define the data type A of graphs equipped with a completeness predicate, we provide the signature

signature Σ				
sorts	`Bool`, `Nat`, `Graph`			
functions	`true`, `false`	:		$\to$ `Bool`
	`zero`	:		$\to$ `Nat`
	`succ`	:	`Nat`	$\to$ `Nat`
	`empty`	:		$\to$ `Graph`
	`connect`	:	`Nat` $\times$ `Nat` $\times$ `Graph`	$\to$ `Graph`
	`complete`	:	`Graph`	$\to$ `Bool`
end signature				

and interpret the symbols in the obvious way:

$$
\begin{array}{lcl}
\texttt{Bool}^A & := & \{\mathit{true}, \mathit{false}\} \\
\texttt{Nat}^A & := & \mathbb{N} \\
\texttt{Graph}^A & := & \{\, G \subset \mathbb{N} \times \mathbb{N} \mid G \text{ finite}\} \\
\texttt{true}^A & := & \mathit{true} \\
\texttt{false}^A & := & \mathit{false} \\
\texttt{zero}^A & := & 0 \\
\texttt{succ}^A(n) & := & n+1 \\
\texttt{empty}^A & := & \emptyset \\
\texttt{connect}^A(n, m, G) & := & G \cup \{(n, m)\} \\
\texttt{complete}^A(G) & := & \begin{cases} \mathit{true} & \text{if } G \text{ is complete} \\ \mathit{false} & \text{otherwise.} \end{cases}
\end{array}
$$

Notice that the data type A is obviously term-generated and computable. The following two sections will establish the existence of a monomorphic specification for A with quantifiers and the non-existence of one without (both not using hidden functions).

5 Specifiability using quantifiers

In order to show that, by using quantifiers, the data type A can be specified without introducing hidden functions, we subsequently give a suitable specification. For convenience, we agree on two abbreviations (but note that we are not adding any new symbols to the signature):

$$\begin{aligned} \mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{g}) &:\equiv (\mathtt{connect}(\mathtt{x},\mathtt{y},\mathtt{g}) = \mathtt{g}) \\ \mathit{in}(\mathtt{x},\mathtt{g}) &:\equiv \exists\, \mathtt{z}.\, (\mathit{connected}(\mathtt{x},\mathtt{z},\mathtt{g}) \vee \mathit{connected}(\mathtt{z},\mathtt{x},\mathtt{g}))\,. \end{aligned}$$

specification COMPL

signature Σ

variables $\mathtt{x},\mathtt{y},\mathtt{u},\mathtt{v} : \mathtt{Nat}$
$\mathtt{g},\mathtt{h} : \mathtt{Graph}$

axioms

(1) $\mathtt{true} \neq \mathtt{false}$

(2) $\mathtt{succ}(\mathtt{x}) \neq \mathtt{zero}$

(3) $\mathtt{succ}(\mathtt{x}) = \mathtt{succ}(\mathtt{y}) \rightarrow \mathtt{x} = \mathtt{y}$

(4) $\neg\, \mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{empty})$

(5) $\mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{connect}(\mathtt{u},\mathtt{v},\mathtt{g}))$
$\quad \leftrightarrow (\mathtt{x} = \mathtt{u} \wedge \mathtt{y} = \mathtt{v}) \vee \mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{g})$

(6) $\mathtt{g} = \mathtt{h} \leftrightarrow \forall\, \mathtt{x},\mathtt{y}.\, (\mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{g}) \leftrightarrow \mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{h}))$

(7) $\mathtt{complete}(\mathtt{g}) = \mathtt{true} \vee \mathtt{complete}(\mathtt{g}) = \mathtt{false}$

(8) $\mathtt{complete}(\mathtt{g}) = \mathtt{true}$
$\quad \leftrightarrow \forall\, \mathtt{x},\mathtt{y}.\, (\mathit{in}(\mathtt{x},\mathtt{g}) \wedge \mathit{in}(\mathtt{y},\mathtt{g}) \rightarrow \mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{g}))$

end specification

Theorem 3. *The first-order specification* COMPL *specifies the data type A, i.e. Mod(COMPL) $= [A]$.*

Proof. We can easily verify that all axioms are valid in A. Since A is term-generated, A is a model of COMPL and it remains to prove the monomorphicity of COMPL.

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 each kind of ground equation separately:

(a) $t_1 = t_2$ for ground terms t_1, t_2 of the sort Nat:
Due to axioms (2) and (3), 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.

(b) $t_1 = t_2$ for ground terms t_1, t_2 of the sort `Graph`:
We first use induction on the structure of t and axioms (4) and (5) to show that for all ground terms t'_1, t'_2 of the sort `Nat` and t of the sort `Graph`, either COMPL $\models$ *connected*(t'_1, t'_2, t) or COMPL $\models \neg$ *connected*(t'_1, t'_2, t).
Then, it immediately follows from axiom (6) that COMPL fixes arbitrary ground equations of the sort `Graph`.

(c) $t_1 = t_2$ for ground terms t_1, t_2 of the sort `Bool`:
Since the atoms on the right hand side of axiom (8) are equations of the sort `Graph`, it follows from (b) that for all ground terms t of the sort `Bool`, either COMPL $\models$ `complete`$(t) =$ `true` or COMPL $\models$ `complete`$(t) \neq$ `true`.
Using axiom (7), we find that either COMPL $\models$ `complete`$(t) =$ `true` or COMPL $\models$ `complete`$(t) =$ `false`. Ground terms `complete`(t) can thus be "reduced" to either `true` or `false`, i.e. all ground equations $t_1 = t_2$ of the sort `Bool` can be "reduced" to equations $t'_1 = t'_2$ with $t'_1, t'_2 \in \{$`true`, `false`$\}$.
The latter are fixed due to axiom (1). ■

6 Non-specifiability without quantifiers

The previous section shows that the abstract data type A can be specified without hidden functions, but using quantifiers. We now turn to the question whether the use of quantifiers is indeed necessary.[5] We will answer this question positively by proving the following

Theorem 4. *There is no quantifier-free specification SP without hidden functions such that $Mod(SP) = [A]$.*

Before presenting the actual proof, we give a short outline. While a "local property"—namely that of a missing edge between two non-isolated vertices (vertices that are part of at least one of the graph's edges)—is sufficient for *non-completeness* of a graph, a graph's *completeness* can only be guaranteed by looking at *all* pairs of its vertices and checking whether an edge exists between them.

However, a finite formula without quantifiers can only make statements about a relation between a bounded number of edges. If the size of a graph exceeds this bound, the specification fails to recognize this graph's completeness. Or, if the graph's completeness is actually recognized, then it is upon conditions that are not sufficient in the general case, and the specification will falsely classify some other graph as complete.

While the basic idea can be described fairly easily, the proof itself turns out to be quite cumbersome. We start with some conventions and definitions.

1. v always denotes a valuation.
2. $t, t_i, t_j, t_0, t_1, t_2$ are terms of the sort `Graph`.
3. x, x_i are variables of the sort `Graph`.

[5] For instance, the equality of graphs (in axiom (6)) could also have been specified *without* quantifiers—at the cost of a more difficult proof of theorem 3.

4. $(t)^{A,v}$ denotes the evaluation of the term t in the algebra A under the valuation v.
5. φ always denotes a disjunction of literals (i.e. of equations or inequations between terms), while ϕ stands for an arbitrary formula.
6. m, n are always natural numbers.

Definition 5. 1. For every m, G_m denotes the complete graph

$$G_m \quad := \quad \bigcup_{i,j=0}^{m} \{(i,j)\},$$

and the algebra A_m is defined to be identical to A except for

$$\texttt{complete}^{A_m}(G_m) \quad = \quad \mathit{false}.$$

2. The *distance* $d(G_1, G_2)$ between two graphs G_1, G_2 is the cardinality of their symmetric set difference, i.e. $d(G_1, G_2) := |(G_1 \setminus G_2) \cup (G_2 \setminus G_1)|$.
3. $\mathrm{connects}(\phi)$ is the number of occurrences of the function symbol `connect` in the formula ϕ; $\mathrm{connects}(t)$ is defined likewise.
4. The *maximum* $\max(\phi, A, v)$ of a formula ϕ under a valuation v in an algebra A is the greatest natural number occurring in the evaluation of `Nat` and `Graph` terms of ϕ under v and A.
5. Two terms t_1, t_2 are *adjacent in* φ if there is a term t in φ such that both t_1 and t_2 are subterms of t, or if a literal $t_1 \neq t_2$ exists in φ.
   $\sim_\varphi$ denotes the transitive, symmetric closure of adjacency and is obviously an equivalence relation. If $t_1 \sim_\varphi t_2$ holds, then t_1 and t_2 are called *connected in* φ.
6. $T_{\varphi,v,m}$ denotes the set of all terms t such that there exists a term t_0 in φ with $t \sim_\varphi t_0$ and $(t_0)^{A,v} = G_m$.
   Note that $t \in T_{\varphi,v,m}$ implies that t appears in φ. Also, $T_{\varphi,v,m}$ is closed under subterms by definition of $\sim_\varphi$. In particular, if a term t contains a variable x, the term t is in $T_{\varphi,v,m}$ iff $x \in T_{\varphi,v,m}$, a fact that we will be using regularly without explicitly mentioning it.
7. For a valuation v, the valuation $v_{\varphi,m,n}$ is defined to be identical to v except that we set $v_{\varphi,m,n}(x) := v(x) \cup \{(n+1, n+2)\}$ for variables $x \in T_{\varphi,v,m}$.

Lemma 6. *Let v be an arbitrary valuation:*

(a) *If t does not contain a variable of the sort* `Graph`*, then*

$$\left|(t)^{A,v}\right| \quad \leq \quad \mathrm{connects}(t).$$

(b) *If $t \in T_{\varphi,v,m}$ and none of the inequations of the sort* `Graph` *in φ is valid under v, then*

$$d\left(G_m, (t)^{A,v}\right) \quad \leq \quad \mathrm{connects}(\varphi).$$

Proof. (a) can be proven by a simple induction on the structure of t.

To prove (b), let t_0 be a term with $t \sim_\varphi t_0$ that is evaluated to G_m under v and A. Such a term t_0 must exist due to the definition of $T_{\varphi,v,m}$.

Let $(t_1, t_2, \ldots, t_{j-1}, t_j)$ be a path of minimal length j connecting t and t_0 in the adjacency relation, i.e. $t_1 = t$ and $t_j = t_0$. Such a path must exist because of $t \sim_\varphi t_0$. For any term t' in φ, there is at most one pair (t_i, t_{i+1}) that is adjacent through being subterms of t'. Otherwise, if $i_1 < i_2$ were indices of two such pairs, we could obtain a path of length $j - (i_2 - i_1)$ by leaving out the terms $t_{i_1+1}, \ldots, t_{i_2}$.

The above statement also implies that for any two pairs (t_{i_1}, t_{i_1+1}), (t_{i_2}, t_{i_2+1}), that are adjacent through being subterms of t'_{i_1} or t'_{i_2}, t'_{i_1} cannot be a subterm of t'_{i_2}, and neither can t'_{i_2} be a subterm of t'_{i_1}. Thus, all t'_i can be chosen to be maximal, i.e. not to be subterms of any other term t' in φ.

To prove the desired inequation, we look at every step i in the path connecting t and t_0, and add up their contributions to the total set difference of the terms' evaluations. If t_i and t_{i+1} are connected through an inequation in φ, then the precondition that none of the `Graph` inequalities hold yields that both terms are evaluated to the same graph, so that this step's contribution is 0.

Otherwise, t_i and t_{i+1} are both subterms of some term t'_i in φ. This implies that their evaluation can differ by no more than $\mathrm{connects}(t'_i)$ edges (which can be proven by induction on the structure of t'_i). Using the above observation about the choice of the terms t'_i, we obtain:

$$
\begin{aligned}
& d\left((t)^{A,v}, (t_0)^{A,v}\right) \\
\leq\; & \sum_{i=1}^{j-1} d\left((t_i)^{A,v}, (t_{i+1})^{A,v}\right) \\
\leq\; & \sum_{t' \in \varphi \text{ maximal}} \mathrm{connects}(t') \\
\leq\; & \mathrm{connects}(\varphi).
\end{aligned}
$$

■

Lemma 7. *Let* $m \geq \mathrm{connects}(\varphi)$, *and* $n \geq \max(\varphi, A, v)$. *For all literals* λ *in* φ *that do not contain a term* `complete`(t) *with* $t \in T_{\varphi,v,m}$, *we assume* $A, v \not\models \lambda$. *Then, for all of these literals, it holds that* $A, v_{\varphi,m,n} \not\models \lambda$.

Proof. We check the different forms that the literals λ can have. The claim is obvious for equations and inequations between terms of the sorts `Nat` or `Bool` (if a literal includes a term `complete`(t), the precondition yields $t \notin T_{\varphi,v,m}$), as well as equations and inequations between terms of the sort `Graph` that do not contain any variables $x \in T_{\varphi,v,m}$. In all of these cases, every such term is evaluated in the same way under $v_{\varphi,m,n}$ and v.

The remaining cases are literals of the form $t_1 = t_2$ or $t_1 \neq t_2$, where at least one of the terms contains a variable $x \in T_{\varphi,v,m}$. If this applies to both terms, it can be easily checked that the valuation $v_{\varphi,m,n}$ adds the same edge $(n+1, n+2)$

to the evaluations of both terms, and that this edge was not previously contained in any of them. Thus, the equation holds under the valuation $v_{\varphi,m,n}$ iff it holds under v.

If both terms contain variables of the sort `Graph`, but only one of them is in $T_{\varphi,v,m}$, the literal has to be an equation (otherwise, the other variable would be in $T_{\varphi,v,m}$ by definition). Since the edge $(n+1, n+2)$ is added to the valuation of the term in $T_{\varphi,v,m}$ under $v_{\varphi,m,n}$, but not to that of the other term, it is clear that the equation cannot hold.

If λ is an equation such that one of the terms does not contain a variable of the sort `Graph`, an argument similar to the one just presented shows the claim. The case that λ is an inequation such that one of the terms does not contain a variable cannot occur, because this would imply $\texttt{empty} \in T_{\varphi,v,m}$, and lemma 6(b) would yield $d(G_m, \emptyset) \leq \text{connects}(\varphi) \leq m$, clearly a contradiction to $|G_m| = (m+1)^2$. ■

Lemma 8. *Let $\varphi = \bigvee_{j=1}^{J} \lambda_j$ such that all literals λ_j of the sort* `Bool` *are either of the form* $\texttt{complete}(t_1) = \texttt{true}$ *or of the form* $\texttt{complete}(t_1) = \texttt{false}$.[6] *Then, for all $m \geq \text{connects}(\varphi)$, we have*

$$A \models \varphi \implies A_m \models \varphi.$$

Proof. We fix an arbitrary $m \geq \text{connects}(\varphi)$. For any valuation v, we fix $n \geq \max(\varphi, A, v)$ and show that if both $A, v \models \varphi$ and $A, v_{\varphi,m,n} \models \varphi$ hold, then $A_m, v \models \varphi$. This clearly implies the lemma.

If there is a $j \in \{1, \ldots, J\}$ with $A, v \models \lambda_j$ such that λ_j is not of the form $\texttt{complete}(t_j) = \texttt{true}$ with $(t_j)^{A,v} = G_m$, it is obvious that $A_m, v \models \lambda_j$, since the only difference between A and A_m is the evaluation of `complete` at G_m. This implies $A_m, v \models \varphi$ and hence our claim.

Otherwise, $A, v \models \lambda_j$ can only hold for literals $\texttt{complete}(t_j) = \texttt{true}$, where $(t_j)^{A,v} = G_m$ (especially implying $t_j \in T_{\varphi,v,m}$). For all other literals, we have $A, v \not\models \lambda_j$. Therefore, we can use lemma 7 to conclude that the only literals with $A, v_{\varphi,m,n} \models \lambda_i$ can be those including a term $\texttt{complete}(t_i)$ with $t_i \in T_{\varphi,v,m}$, and because of $A, v_{\varphi,m,n} \models \varphi$, there must be at least one such $i \in \{1, \ldots, J\}$.

t_i must contain a variable of the sort `Graph`, because otherwise, we could use lemma 6(a) to conclude that the evaluation of t_i has at most m edges, in contradiction to t_i being evaluated to G_m (which has $(m+1)^2$ edges). We can therefore use the definition of $v_{\varphi,m,n}$ and induction on the structure of t_i to show that the evaluation of t_i under $v_{\varphi,m,n}$ includes the edge $(n+1, n+2)$, but not the edge $(n+1, n+1)$—thus, it is not a complete graph. Hence, $A, v_{\varphi,m,n} \models \lambda_i$ implies that λ_i is of the form $\texttt{complete}(t_i) = \texttt{false}$ (here, we use the restriction placed on literals of the sort `Bool`).

We now look at the evaluation of t_i under v, writing $G := (t_i)^{A,v}$. If $G = G_m$, we immediately have $A_m, v \models \lambda_i$ because of A_m's definition. Otherwise, lemma 6(b) states that G and G_m differ by at most $\text{connects}(\varphi) \leq m$ edges. Since

[6] This assumption is only used to make the proof less technical. The lemma also holds without the assumption.

any complete graph other than G_m differs from G_m by at least $2m + 1$ edges (which can be proven quite easily), G cannot be complete, and hence $A_m, v \models \lambda_i$, completing our proof. ■

Proof of theorem 4. Using lemma 8, the proof of theorem 4 is quite straightforward. If SP is any quantifier-free specification without auxiliary functions, and Φ is the set of its axioms, we form the conjunction of all axioms in Φ. We then replace every occurrence of any equation $t_1^B = t_2^B$ (with terms t_1^B, t_2^B of the sort `Bool`) by

$$t_1^B = \texttt{true} \quad \leftrightarrow \quad t_2^B = \texttt{true},$$

and similarly any inequation $t_1^B \neq t_2^B$ by

$$t_1^B = \texttt{true} \quad \leftrightarrow \quad t_2^B = \texttt{false},$$

Transforming the resulting formula into conjunctive normal form and replacing inequations $t^B \neq \texttt{true}$ (resp. $t^B \neq \texttt{false}$) by $t^B = \texttt{false}$ (resp. $t^B = \texttt{true}$) yields a formula $\phi = \bigwedge_{k=1}^{K} \varphi_k$ such that each φ_k is a disjunction of literals satisfying the requirements of lemma 8.

Clearly, $A \in \mathit{Mod}(SP)$ implies $A \models \phi$. We choose some $m \geq \text{connects}(\varphi_k)$ for all $k \in \{1, \ldots, K\}$. The lemma then implies $A_m \models \varphi_k$ for all k, and therefore we have $A_m \models \phi$. This implies $A_m \models \Phi$, and hence $A_m \in \mathit{Mod}(SP)$. Thus $\mathit{Mod}(SP) \neq [A]$. ■

7 Conclusions

It is well known that there is no need for introducing quantifiers in order to specify computable data types if hiding facilities are used. Matters are quite different if hiding facilities are disabled: in this paper, we provided an example of a (natural) computable data type that has a monomorphic first-order specification but no monomorphic quantifier-free specification (both without hidden symbols).

Even if the use of quantifiers does not completely compensate the loss of expressiveness caused by disabling hiding (cf. [Ber93, Sch97a, Sch97b]), theorem 1 indicates that quantifiers make up for this loss at least to some extent. In other words, we have shown that the use of quantifiers can help to avoid introducing hidden functions. Since hidden functions often make specifications less readable and may cause technical inconvenience (e.g. the handling of clashes between hidden symbols when combining specifications), this paper can be seen as support for the provision of quantifiers in specification languages.[7]

While we believe that our result is proven for the first time, the problem was already mentioned in [BDP+79]: Broy et. al. give several examples for specifications that demonstrate the usefulness of quantifiers, and they even claim:

[7] Notice that our arguments are also valid for the case that initial algebra semantics is adopted: Since there are no homomorphisms from A to the non-standard models A_m considered in section 6, the restriction to initial models does not eliminate the necessity of either quantifiers or hidden functions. (Unfortunately, quantifiers would spoil the guarantee of the existence of initial models and executability.)

> "A formulation without existential quantifiers requires the introduction of additional (hidden) functions, which makes the presentation inconvenient ..."

However, they refer to a non-monomorphic data type[8] and do not provide a proof for their statement.

Furthermore, our work is somewhat related to the work done in finite model theory and descriptive complexity, where especially the (first-order) definability of properties of finite graphs is studied (cf. e.g. [EF95] and [Imm82]). A typical result is, for instance, that there is no first-order formula φ such that

$$G \models \varphi \iff G \text{ is connected}$$

for all finite graphs G (where the variables in φ are required to range over the nodes of G) (cf. [Gai82]). Unfortunately, such results do not carry over to our setting because we can use variables ranging over graphs (which allows, for instance, a first-order specification of connectedness) and even inductive definitions (since the property to be specified is named by a symbol, in our case `complete`).

Moreover, instead of merely providing an example, one could be interested in a *characterization* of the data types that can or cannot be specified without hiding and without quantifiers. It is well known that for any first-order formula φ there is an equivalent quantifier-free formula φ' iff the class of models of φ is closed under subalgebras [Rob74, Theorem 3.3.4]. However, this result does not apply in our context, because even if the class of *all* models of φ is not closed under subalgebras, there can be a quantifier-free formula having the same *term-generated* models.

Another approach is pursued in [BT87], [BBTW81], and others: the authors characterize specifiability (using hidden functions) in terms of complexity (cf. e.g. [EM81, AT82]). However, fact 2 indicates that specifiability without hidden functions generally does not imply any bound on the data type's complexity. Conversely, it might be interesting to study whether (certain) complexity bounds guarantee specifiability without hidden functions.

Acknowledgments. We thank the anonymous referees for their helpful comments.

References

[AT82] P.R.J. Asveld and J.V. Tucker. Complexity theory and the operational structure of algebraic programming systems. *Acta Inform.*, 17:451–476, 1982.

[BBTW81] J.A. Bergstra, M. Broy, J.V. Tucker, and M. Wirsing. On the power of algebraic specifications. In J. Gruska and M. Chytil, editors, *Proc. of the 10th Symp. on Math. Found. of Comp. Sci.*, volume 118 of *LNCS*, pages 193–204. Springer, 1981.

[8] In the case of non-monomorphic data types, it is much easier to construct an example that demonstrates the necessity of quantifiers (e.g. [BDP+79, p. 84]).

[BDP+79] M. Broy, W. Dosch, H. Partsch, P. Pepper, and M. Wirsing. Existential quantifiers in abstract data types. In H.A. Maurer, editor, *Automata, Languages and Programming, 6th Colloquium*, volume 71 of *LNCS*, pages 73–87. Springer, 1979.

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

[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.

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

[EM81] H. Ehrig and B. Mahr. Complexity of algebraic implementations for abstract data types. *Computer and System Sciences*, 23(2):223–253, 1981.

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

[Imm82] N. Immerman. Upper and lower bounds for first order expressibility. *Computer and System Sciences*, 25:76–98, 1982.

[Kem98] D. Kempe. Ausdrucksmächtigkeit von Quantoren in algebraischen Spezifikationen. Master's thesis, Universität Karlsruhe, August 1998.

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

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

[Mat93] Y. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, 1993.

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

[Rob74] A. Robinson. *Introduction to model theory and to the metamathematics of algebra*. Studies in Logic and the Foundations of Mathematics. North Holland Publishers, 1974.

[Sch97a] A. Schönegge. An answer to the hidden function question for algebraic specification methods (abstract), 4th Workshop on Logic, Language, Information and Computation. *Logic Journal of the IGPL*, 5(6), 1997.

[Sch97b] A. Schönegge. The hidden function question revisited. In *Proc. of the 6th Int. Conf. on Algebraic Methodology and Software Technology (AMAST '97)*, volume 1349 of *LNCS*, pages 451–464. Springer, 1997.

[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 Programming Languages and Systems*, volume 4, pages 711–732, 1982.

[Wir90] M. Wirsing. *Algebraic Specification*, volume B of *Handbook of Theoretical Computer Science*, pages 675–788. Elsevier Science Publishers B. V., 1990.