# Relationships Between Equational and Inductive Data Types

Eric G. Wagner

Wagner Mathematics, 1058 Old Albany Post Road,
Garrison, NY 10524, USA
wagner@highlands.com

**Abstract.** This paper explores the relationship between equational algebraic specifications (using initial algebra semantics) and specifications based on simple inductive types (least fixed points of equations using just products and coproducts, e.g. $N \cong 1 + N$). The main result is a proof that computable data type (one in which the corresponding algebra is computable in the sense of Mal'cev) can be specified inductively. This extends an earlier result of Bergstra and Tucker showing that any computable data type can be specified equationally.

## 1 Introduction

For the purposes of this paper, an equational data type is one specified as the initial $\Sigma$-algebra for a given signature, $\Sigma$ and set of equations $E$ over $\Sigma$; and a simple inductive data type is one defined as the least fixpoint solution of equations using just products and coproducts, e.g., $N \cong 1 + N$.

The early papers on equational data type specifications [5–8], and inductive specifications [12], were written in the seventies. (See [11] for other earlier references relevant to inductive types). The two approaches seem rather different:

1. Early papers on inductive data types used $\rightarrow$ in addition to $+$ and $\times$ and thus introduced higher level types not found in the equational approach.
2. In the equational approach it is easy to specify types such as finite sets of natural numbers, but this can not be done directly in the inductive approach, that is, finite sets do not appear as a fixpoint of equations using just $+$ and $\times$.
3. Papers on inductive types frequently also consider co-inductive types: greatest fixpoints of equations in $+$ and $\times$ (and $\rightarrow$). This also leads to data types, such as possibly-infinite strings and possibly-infinite trees; I do not know of any equational specifications for such types.
4. Equational specifications are not necessarily implementable as is shown by the existence of unsolvable word problems; but simple inductive data types are always implementable [14].

We could continue listing differences, but the real point of this paper is to argue that, for all "practical purposes", the equational and inductive approaches are equally powerful. Of course this is only true when we make certain "practical" restrictions:

H.-J. Kreowski et al. (Eds.): Formal Methods (Ehrig Festschrift), LNCS 3393, pp. 259–274, 2005.

1. We restrict ourselves to computable data types.
2. We restrict ourselves to simple inductive types (no $\rightarrow$, no greatest fixpoints).

The first restriction is justified on the grounds that, after all, we do want to use the specified data types in real programs which means they must be implementable, i.e., computable. This does not justify the second restriction, but without the second restriction we are clearly faced with insurmountable differences.

Section 2 of the paper reviews the relevant concepts from the theories of equational specifications and recursive functions, and the work on computable data types (or algebras) from [13] and [1]. Section 3 explores simple inductive specifications and ways to enrich them with additional operations. In particular, it is shown how we can generalize primitive recursion and the While-do operator to inductive types. Section 4 applies the results of Section 3 to traditional recursion theory; showing that both primitive recursion and minimalization can be expressed using While-do. The main result is given in Section 5, where we show that computable data types can be specified both equationally and inductively following a particular specification strategy.

## 2 Preliminaries

### 2.1 Signatures and Algebras

The following is meant only to clarify our notation, we assume the reader is already familiar with the concepts.

**Definition 1.** *A* signature *consists of the following data*

$S$*, a finite set, the* set of sorts.
$F$*, a finite set, the* set of operators.
$\mathrm{dom} : F \rightarrow S^*$
$\mathrm{cod} : F \rightarrow S$

*We write* $\Sigma = \lceil S, F, \mathrm{dom}, \mathrm{cod} \rceil$ *to indicate that* $\Sigma$ *is a signature with the given data.*

*Given a signature* $\Sigma = \lceil S, F, \mathrm{dom}, \mathrm{cod} \rceil$ *we define a* $\Sigma$-algebra*,* $A$*, as consisting of*

1. *For each* $s \in S$ *a set* $A(s)$*, called the* carrier of sort $s$*.*
2. *For each* $\sigma \in F$ *a function* $A(\sigma) : A(\mathrm{dom}(\sigma)) \rightarrow \mathrm{cod}(\sigma)$ *where, for any* $w \in S^*$*, if* $w = s_1 \cdots s_n$*, then* $A(w) = A(s_1) \times \cdots \times A(s_n)$*.*

*We will sometimes denote a* $\Sigma$*-algebra as* $A = \lceil \langle A(s) \mid s \in S \rangle, \langle A(\sigma) \mid \sigma \in F \rangle \rceil$*.*

*If* $A$ *and* $B$ *are* $\Sigma$*-algebras then a* $\Sigma$-homomorphism*,* $h : A \rightarrow B$*, consists of an* $S$*-indexed family of mappings,* $h = \langle h(s) : A(s) \rightarrow B(s) \mid s \in S \rangle$*, such that for any* $\sigma \in F$*,*

$$B(\sigma) \circ h(\mathrm{dom}(\sigma)) = h(\mathrm{cod}(\sigma)) \circ A(\sigma),$$

*where, for any* $w \in S^*$*, if* $w = s_1 \cdots s_n$ *then* $h(w) = h(s_1) \times \cdots \times h(s_n)$*.*