Algebraic methods in semantics

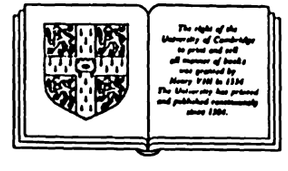
Edited by

MAURICE NIVAT

Professor of Information Science, University of Paris VII

JOHN C. REYNOLDS

Professor of Computer and Information Science
Syracuse University, New York



CAMBRIDGE UNIVERSITY PRESS

Cambridge

London New York New Rochelle

Melbourne Sydney

14

Initiality, induction, and computability†

JOSÉ MESEGUER, JOSEPH A. GOGUEN

† Research supported in part by Office of Naval Research Contracts N00014-80-0296 and N00014-82-C-0333, and National Science Foundation Grant

1 Introduction

This paper surveys, unifies, and extends a number of results on induction and computability in the context of an algebraic approach to the semantics of programming. The close relationship between computability, induction, and initiality is emphasized.

Highlights of this paper include: a software engineering motivation for the initial algebra approach to data abstraction; a review of many-sorted general algebra, including rules for equational deduction that are sound and complete; simple 'inductive' characterizations of initiality (including generalized Peano axioms); constructions for both initial and final (i.e., minimal) algebra realizations of abstract software modules; and a detailed introduction to computable algebras and their relationship to initiality, finality, and rewrite rules, showing in particular how Gödel numberings arise from initiality, and how equationally defined equality relates to both theorem proving by 'inductionless induction' and computability. The latter permits us to give a purely algebraic characterization of computable algebras: an algebra is computable iff it is a reduct of an initial model of a finite equational equality presentation.

2 Abstract data types and programming methodology

Data abstraction enjoys considerable popularity. It is widely recognized as an important technique for structuring programs, perhaps even more useful than structuring programs by flow of control as in traditional flow charts or more modern data flow diagrams. Data abstraction has been advocated, for example, by Jackson [48] (though he

does not use the phrase 'data abstraction'), and is discussed more formally by Guttag [38], Liskov & Zilles [62], Goguen, Thatcher, Wagner & Wright [37] and many others. The basic idea seems to have first surfaced in the 'class' concept of the Simula programming language of Dahl, Myhrhaug & Nygaard [21]. Similar constructs appear in many later languages with names like 'form', 'module', 'cluster', 'object', 'capsule', 'domain', 'package', 'type', 'bundle', and even 'category'. (The names of the languages involved are left as an exercise to the interested reader.) This paper favors the more generic term *module*.

2.1 *What is data abstraction?*

The methodological use of ADTs in programming is to suggest (or better, to require) grouping together in one module all the basic functions that manipulate one (or more) sort of data, and then 'hiding' the representation, in the sense that only the functions defined in that module can actually see the representation. Thus, any agent wishing to manipulate this data can only do so by calling the functions provided by the module. This technique is called *data encapsulation*, and such a module is often called a *data abstraction*; the advantage is to localize the effects of changing representation in an especially clear and simple way. It is often claimed (and we also believe) that this approach facilitates reading, writing, specifying, designing, modifying, maintaining, and reasoning about programs.

2.2 *Abstract machines*

Much confusion can be avoided by distinguishing sharply between (concrete or abstract) *data types* that are just algebras, and (concrete or abstract) *machines* that in addition may have internal *states*.† A typical data type is that of the *integers*; there are certain values, namely integers, and certain operations upon them. While it is possible and fruitful to investigate abstract machines with the techniques of abstract data types, quite different concepts and techniques are also important, such as reachability, observability, and minimality. Abstract data types are

† This distinction is sometimes indicated with the words 'immutable' and 'mutable', indicating that data items (such as the integer 3) are ideal and 'timeless', while the behavior of a machine can vary with time; see [33] for a formal treatment.

useful for understanding the type systems of programming languages, especially when they permit user-defined types as in ALGOL 68 [87], and for deciding the correctness of data representations. Abstract machines are useful for understanding the specification and implementation of software modules, in roughly the sense of Parnas [74], and as used in the HDM methodology [61] and in CLU [63]. The fact that many common examples can be viewed from either perspective contributes to the potential confusion.

There is, for instance, the rather pointless controversy about whether final or initial algebra semantics is 'best'. For abstract machines, it is *behaviour* that matters. Machines that represent and manipulate their internal states differently (i.e., are nonisomorphic as data types) can still have the same behavior [26]. A software module can usually be *realized* in many different ways; among these, the *final* one uses as little storage as possible for internal states, while the *initial* one has no sharing at all for storage of states [33]. Because the space efficiency of the final realization can greatly reduce its time efficiency, there are many cases where the pragmatically correct choice of data representation is neither initial nor final, but rather something between. A good example is the list structure of LISP, which is often implemented by giving a unique cell to each atom, but not to each list structure. QLISP can be seen as an experiment with the 'hashcons' final realization of list structure; it was found to run too slowly for many applications. Abstract Prolog machines also now tend to replicate information rather than to share it, and are therefore closer to an initial realization than to a final realization [90]. In summary, initial realizations are appropriate in case all behavior is visible behavior; final realizations may be appropriate in case there are hidden internal states, but often the most practical realizations, although neither initial nor final, are rather close to being initial.

2.3 *What good is algebra?*

The basic argument for an algebraic approach is both simple and compelling. A software module has *exactly* the same structure as an algebra: the various sorts of data involved (including states, if any) form sets, and the operations of interest are functions among these sets (Section 3 precisely defines this sense of algebra). This argument is reinforced by the powerful, general and appropriate tools that modern algebra provides. It is also reinforced by the remarkable fact (to be discussed later in this paper, together with other important characterizations) that any *computable* algebra can be specified as an initial algebra for a *finite* number of equations (after adding perhaps a finite number of auxiliary functions); this shows the general applicability of finitary algebraic specifications in computer science.

Three different algebraic approaches to ADTs emerged in 1975: Zilles [91] gave an abstract of some results to appear in his Ph.D. thesis; Guttag [38] completed his dissertation; and ADJ (Goguen, Thatcher, Wagner & Wright [37]) sketched their initial algebra approach. Zilles [92] suggested a new kind of algebra, called 'data algebra', based on the notation of Cohn [20]; Zilles' use of free algebras essentially corresponds to ADJ's use of initiality. Guttag's work introduced the important ideas of 'consistency' and 'sufficient completeness' later formalized by others, and opened up the study of modules with states, i.e., abstract machines. ADJ, using initiality and the algebraic notation of Goguen [29], were able to formalize the ADT concepts entirely rigorously within standard many-sorted general algebra. Strangely enough, Zilles' work, though motivated by CLU 'clusters', actually treated ADTs rather than abstract machines. Each of these three approaches has subsequently been followed up by its originators, as well as by many others, and today there is a vast literature on the subject. Parnas [74], Milner [71] and Hoare [41] were important early theoretical influences in this development. All three approaches recognize that a concrete data type is a many-sorted algebra.† The algebraic approach to *abstract* data types (any of the three versions) goes beyond this in that one gives some *equations* that the functions ought to satisfy, and then restricts attention to models where they do. The initial algebra approach produces a 'standard' model that is defined uniquely up to renaming data items, namely 'the' initial algebra for the given signature and equations. What is magic about this is that a set of equations that the operations are supposed to satisfy actually *defines* the data; there is no need to talk about how the data is represented.

There are many different ways to precisely define data abstraction (see Section 4); a fairly simple one (from Burstall & Goguen [18]) is as follows: assume that we are given a concrete data type and that we can tell whether or not two concrete data items in it represent the same abstract data item; call the two concrete data items *equivalent* in that case. (Thus, an abstract data item is an equivalence class of concrete data items; for example, 1, 01, and 001 are three different concrete data items representing the same abstract data item, namely the abstract integer 'one'.) Given a signature of symbols for operations and constants, and a set of equations using the

† It is not clear where this insight originated. The earliest work we know is Goguen [27].

symbols from it, call a data representation *standard* if and only if it has the following two properties:

1. *No junk:* Every data item can be constructed using only the constants and operations in the signature. (A data item that cannot be so constructed is 'junk'.)
2. *No confusion:* Two data items are equivalent if and only if they can be proved equal using the equations. (Two data items that are equivalent but cannot be proved so from the given equations are said to be 'confused'.)

Section 4 shows that these two conditions define an algebra *uniquely* up to renaming of its data items. It also shows that 'no junk' is equivalent to structural induction over the signature, and that the two conditions together are equivalent to the 'unique homomorphism' condition usually called 'initiality'. Thus, a model is initial if and only if it has the minimal number of data items (none that cannot be constructed from those that are given) and satisfies the minimal number of ground equations (none that do not follow from those that are given).

3 Many-sorted algebra

So-called 'general' (or 'universal') algebra was established by Birkhoff [15] in order to subsume many basic aspects of particular algebraic systems into a single framework. This work involved only one sort of data, proving the existence of initial algebras as well as giving many other basic results. It was later generalized to many sorts by Higgins [39], by Birkhoff & Lipson [16], and by Benabou [2] following the more abstract approach of Lawvere [59].

A simpler notation for many-sorted algebra that is now often used in computer science was introduced by Goguen [29]. It uses indexed (or sorted) sets, defined as follows: let S be a set (of *sorts*), then an *S-indexed* (or *S-sorted*) *set* A is just a family of component sets A_s for each index s in S. If A and B are both S-indexed sets, then a *mapping of S-indexed sets* (also called an *S-sorted function*) $f: A \to B$ is an S-indexed family of functions

$$\langle f_s: A_s \to B_s \mid s \text{ in } S \rangle.$$

We now apply this to many-sorted general algebra: An S-sorted *signature* Σ is an $S^* \times S$-sorted family $\langle \Sigma_{w,s} \mid w \text{ in } S^*, s \text{ in } S \rangle$; σ in $\Sigma_{w,s}$ is a function symbol of *arity* w and *sort* s; the arity of a function symbol expresses what sorts of data it expects to see as inputs and in what order; and the sort of a function symbol expresses the sort of data it returns. A constant symbol of

sort s has arity the empty string λ; i.e., it is a member of $\Sigma_{\lambda,s}$. Signatures formalize the notion of a (strongly typed) collection of functions available to the user of an abstraction. For example, we might have $S = \{$**stack, nat, bool**$\}$ $\Sigma_{\text{stack,stack}} = \{\text{POP}\}$, $\Sigma_{\lambda,\text{stack}} = \{\text{EMPTY}\}$, $\Sigma_{\text{stack nat,stack}} = \{\text{PUSH}\}$ and so on, for a stack-of-naturals abstraction.

Then a *Σ-algebra* A consists of an S-indexed family $\langle A_s \mid s \text{ in } S \rangle$ of *carrier* sets, and for each function symbol σ in $\Sigma_{w,s}$ an actual function $\alpha(\sigma): A^w \to A_s$ where $A^w = A_{s1} \times \cdots \times A_{sn}$ when $w = s1 \ldots sn$ (when $w = \lambda$, then A^w is a one-point set). Notice that α is an $S^* \times S$-indexed family

$$\alpha_{w,s}: \Sigma_{w,s} \to [A^w \to A_s]$$

of *interpretation* mappings for the function symbols in Σ, each $\alpha_{w,s}$ interpreting σ in $\Sigma_{w,s}$ as a function to A_s from A^w. (Here $[A \to B]$ denotes the set of all functions from A to B.) It is usual to write σ for $\alpha(\sigma)$ if the algebra in question is clear from context, and it is often convenient to write σ_A if it is not.

According to current practice in abstract algebra, one should define not just some structure of interest, but also *functions* that *preserve* that structure. We do this as follows: a *Σ-homomorphism* from a Σ-algebra A to another B is an S-indexed function $f: A \to B$ that 'preserves the function symbols in Σ' in the sense that

$$f(\sigma(a1, \ldots, an)) = \sigma(f(a1), \ldots, f(an)),$$

or more precisely, that

$$f_s(\alpha(\sigma)(a1, \ldots, an)) = \beta(\sigma)(f_{s1}(a1), \ldots, f_{sn}(an)),$$

where β is the interpretation mapping for B, where $w = s1 \ldots sn$, for ai in A_{si} and σ in $\Sigma_{w,s}$. For constants, i.e., for $w = \lambda$, the condition becomes

$$f_s(\alpha(\sigma)) = \beta(\sigma).$$

These equations are called the *homomorphism condition.*

We can now define the central concept of this paper.

Definition 1. A Σ-algebra A is *initial* in a class $\mathscr{C}$ of Σ-algebras if and only if A belongs to $\mathscr{C}$ and for each Σ-algebra C in $\mathscr{C}$ there is one and only Σ-homomorphism from A to C. □

One common case is that $\mathscr{C}$ is the class of all Σ-algebras; another is that $\mathscr{C}$ is the class of all Σ-algebras that satisfy some set E of equations; then $\mathscr{C}$ is called the *variety* of E. In general, the class $\mathscr{C}$ will not be mentioned when it is clear from context.

Perhaps the most basic fact about initial algebras is that any two are 'abstractly the same', in that they differ only in the representations given to

their elements. This is formalized using the following concept: A Σ-*isomorphism* is a Σ-homomorphism f such that each component function f_s is bijective. Then isomorphic Σ-algebras are 'abstractly the same'; this is the essence of the word *abstract* in both the phrases 'abstract algebra' and 'abstract data type'. Indeed, one of the main ideas of abstract algebra is to study algebras (such as groups and vector spaces) independently of how their elements happen to be represented. The following states this basic property of initial algebras; it can be proved using the properties given in Proposition 3 below.

Proposition 2. Let A be initial in a class $\mathscr{C}$ of Σ-algebras; then an algebra A' is initial in $\mathscr{C}$ iff A and A' are Σ-isomorphic. In fact, there is then a unique isomorphism from A to A'. □

The existence of initial algebras is discussed in the next section. We now turn to some basic properties of homomorphisms.

Proposition 3. Let Σ be an S-sorted signature, and let A, B, and C be Σ-algebras.

(1) Given Σ-homomorphisms $f: A \to B$ and $g: B \to C$, their *composition*, $g \circ f: A \to C$ defined by $(g \circ f)_s = (g_s \circ f_s)$, is also a Σ-homomorphism.
(2) The S-indexed function 1_B defined by $(1_B)_s = 1_{B_s}$ is a Σ-homomorphism $B \to B$ called the *identity* at B; moreover, $1_B \circ f = f$ and $g \circ 1_B = g$ whenever these compositions are defined. (The notation id_B may also be used occasionally.)
(3) Given $f: A \to B$ and $g: B \to A$, if $g \circ f = 1_A$ then g is surjective and f is injective.
(4) A Σ-homomorphism $f: A \to B$ is a Σ-isomorphism if and only if there is another Σ-homomorphism $g: B \to A$ such that $f \circ g = 1_B$ and $f \circ g = 1_A$; this g is unique if it exists, and is called the *inverse* of f, denoted f^{-1}. Moreover, $(f^{-1})_s = (f_s)^{-1}$, the indexed family of inverse functions (i.e., converse relations) to f_s for each s in S.
(5) If $f \circ g$ is surjective, then f is surjective.
(6) If $f \circ g$ is injective, then g is injective.

Proof. These are left as exercises in the use of the definitions; the arguments are generally set-theoretic. □

A Σ-*subalgebra* B of a Σ-algebra A is an S-indexed family of subsets $\langle B_s \rangle = B \subseteq A$ that is closed under the operations in Σ, i.e., such that for any σ in $\Sigma_{w,s}$ with $w = s1 \ldots sn$, $\sigma(a1, \ldots, an)$ is in B_s if $ai \in B_{si}$ for $i = 1, \ldots, n$. A Σ-subalgebra of A is essentially the same thing as an injective Σ-homomorphism $g: B \to A$, since such a B is isomorphic to its image in A. A subalgebra of A is *proper* if it is not equal to A; this corresponds to an injective homomorphism that is not an isomorphism.

Proposition 4. If A is initial in a class $\mathscr{C}$ of Σ-algebras, then A has no proper subalgebras in $\mathscr{C}$.

Proof. Assume that P is in $\mathscr{C}$ and $j: P \to A$ is an injective Σ-homomorphism. By initiality of A, there is also a homomorphism $h: A \to P$. Then $j \circ h: A \to A$ is also a Σ-homomorphism, and since there is only one such from A to A, namely the identity on A, we have $j \circ h = 1_A$. By (3) of Proposition 3 this implies that j is surjective, therefore bijective, and thus not proper. □

If $f: A \to B$ is a Σ-homomorphism and C is a Σ-subalgebra of B, then $f^{-1}(C)$, the *inverse image* of C under f, defined by $(f^{-1}(C))_s = \{a \text{ in } A_s \mid f_s(a) \text{ in } C_s\}$, is a Σ-subalgebra of A. Notice that if C is proper and f is surjective, then $f^{-1}(C)$ is also proper.

3.1 *Quotients*

This subsection says everything you always wanted to know about quotients; moreover, it *proves* that it has told you everything in the sense that the properties given actually *characterize* quotients.

The quotient of a set A by an equivalence relation Q on A (identifying some elements of A with others) is formed by considering the Q-equivalence classes (the sets of mutually Q-equivalent elements of A) as the elements of a new set A/Q. Now suppose that A is a Σ-algebra. Unless applying an operation in Σ to equivalent elements of A yields elements that are again equivalent, A's Σ-algebra structure cannot be inherited by A/Q. This motivates the following discussion.

Once again the notation of indexed sets makes it easy to go from the one-sorted case to the many-sorted case. Let A be an S-indexed set. Then an (S-indexed, or S-sorted) *equivalence relation* on A is just an S-indexed family $\{Q_s \mid s \in S\}$, where Q_s is an equivalence relation on A_s (i.e.,

$Q_s \subseteq A_s \times A_s$ is a reflexive, transitive and symmetric relation). If, in addition, A is a Σ-algebra, where Σ is an S-sorted signature, then we call Q a *Σ-congruence relation* provided that the following *substitutivity condition* holds:

for each $\sigma \in \Sigma_{w,s}$ with $w = s1 \ldots sn$, and each $ai, a'i \in A_{si}$ if $aiQ_{si}a'i$, then $\sigma(a1, \ldots, an)Q_s\sigma(a'1, \ldots, a'n)$.

For example, suppose that $S = \{\textbf{nat}, \textbf{bool}\}$, $A_{\textbf{nat}} = \omega$ (the natural numbers) and $A_{\textbf{bool}} = \{T, F\}$; suppose also that $\Sigma_{\lambda,\textbf{bool}} = \{T, F\}$, $\Sigma_{\lambda,\textbf{nat}} = \{0\}$, $\Sigma_{\textbf{nat},\textbf{nat}} = \{\text{inc}\}$, $\Sigma_{\textbf{nat nat},\textbf{bool}} = \{\text{odd}\}$, and $\Sigma_{w,s} = \varnothing$ otherwise. Finally, suppose that these operation symbols have their usual interpretations in A, with $\text{odd}(n) = T$ if n is odd and $\text{odd}(n) = F$ if n is even. Now define an S-sorted relation $Q8$ on A by: $nQ8_{\textbf{nat}}n'$ iff $n - n'$ is divisible by 8; and $bQ8_{\textbf{bool}}b'$ iff $b = b'$. Then the reader may verify that this is indeed a Σ-congruence on A. Another Σ-congruence on A, $Q2$, is given by $nQ2_{\textbf{nat}}n'$ iff $n - n'$ is divisible by 2, with $Q2_{\textbf{bool}} = Q8_{\textbf{bool}}$.

A very general way that congruences arise is given by the following.

Proposition 5. Let Σ be an S-sorted signature, let A and B be Σ-algebras, and let $h: A \to B$ be a Σ-homomorphism. Then defining $aQ_{h,s}a'$ iff $h_s(a) = h_s(a')$, for each $s \in S$ and $a, a' \in A_s$, yields a Σ-congruence on A, denoted Q_h and called the *kernel* of h.

Proof. Substitutivity follows from the homomorphism condition for h: $h_{si}(ai) = h_{si}(a'i)$ implies $\sigma(h_{s1}(a1), \ldots, h_{sn}(an)) = \sigma(h_{s1}(a'1), \ldots, h_{sn}(a'n))$ and therefore $h_s(\sigma(a1, \ldots, an)) = h_s(\sigma(a'1, \ldots, a'n))$. □

We now introduce the quotient algebra construction. Given a Σ-algebra A and a Σ-congruence Q on A, define A/Q to have carriers $(A/Q)_s = A_s/Q_s$ for $s \in S$. Now letting $[a]$ denote the Q-equivalence class of a in A, define the effect of $\sigma \in \Sigma_{w,s}$ on A/Q by $\sigma([a1], \ldots, [an]) = [\sigma(a1, \ldots, an)]$.

Substitutivity guarantees that this is well-defined: if $[ai] = [a'1]$ for $i = 1, \ldots, n$, then $[\sigma(a1, \ldots, an)] = [\sigma(a'1, \ldots, a'n)]$. Therefore A/Q is a Σ-algebra.

For example, using $S = \{\textbf{nat}, \textbf{bool}\}$ with Σ and A as above, $A/Q8$ is the natural numbers modulo 8, and $A/Q2$ is the natural numbers modulo 2, each with an oddness predicate.

Given a Σ-algebra A with a Σ-congruence Q on it, there is a natural *quotient* Σ-homomorphism $q: A \to A/Q$ defined to send $a \in A_s$ to $[a]$ in $(A/Q)_s$. Substitutivity gives the homomorphism property of q. Notice that q is surjective.

We are now ready to say everything (about the quotient construction). The following says, intuitively, that a quotient, a surjective homomorphism, and a function satisfying two certain properties (called 'universal'), are three different ways of looking at the same situation.

Proposition 6. Let $f: A \to A'$ be a Σ-homomorphism. Then the following are equivalent properties of f:

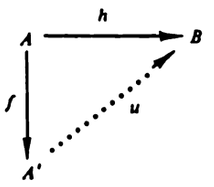
(1) There is an isomorphism $u: A/Q_f \to A'$ such that $u \circ q = f$, for $q: A \to A/Q_f$ the natural quotient function (sending a to $[a]$).
(2) f is surjective.
(3) If $h: A \to B$ is a Σ-homomorphism, then
   a. There exists a function $u: A' \to B$ such that $u \circ f = h$ (i.e., the diagram of Figure 14.1 commutes) iff $Q_f \subseteq Q_h$.†
   b. If such a function u exists, then it is unique, and is a Σ-homomorphism.

Proof. We first show that (1) implies (2): since q is surjective and u is an isomorphism, f is also surjective.

We next show that (2) implies (3a): first suppose that $Q_f \subseteq Q_h$. Then we can define a function $u: A' \to B$ such that the diagram of Figure 1 commutes by $u(a') = h(a)$, where $f(a) = a'$; such an a exists since f is surjective. This is well-defined, because $f(a1) = f(a2) = a'$, implies $h(a1) = h(a2)$, so that $u(f(a1)) = u(f(a2))$. Conversely, if there is a function u such that Figure 1 commutes, then we have that $f(a1) = f(a2)$ implies $u(f(a1)) = u(f(a2))$ implies $h(a1) = h(a2)$, i.e., $Q_f \subseteq Q_h$.

Assuming (2), we now show (3*b*), i.e., that u satisfying (3*a*) is unique, and is a Σ-homomorphism. First uniqueness. If $u': A' \to B$ with $u' \circ f = h$, then $u(f(a)) = h(a) = u'(f(a))$ for each $f(a)$ in A'.

Fig. 14.1. Universal property of the quotient



† The general notion of S-indexed sets tells us that this means $(Q_f)_s \subseteq (Q_h)_s$ for each s in S.

Next we show the homomorphism property for u. For constants this is clear (since h and f are homomorphisms) from commutativity of the diagram. Suppose that σ is in $\Sigma_{w,s}$ with $w = s1 \ldots sn$; then what we have to show is that $u(\sigma(a1', \ldots, an')) = \sigma(u(a1'), \ldots, u(an'))$ for $ai' \in A'_{si}$. Now, let $u(ai')$ be given by $h(ai)$, where ai is such that $f(ai) = ai'$. Further, let $a' = \sigma(a1', \ldots, an')$ and let $a = \sigma(a1, \ldots, an)$. Then $f(a) = \sigma(f(a1), \ldots, f(an))$ because f is a homomorphism, so $f(a) = \sigma(a1', \ldots, an') = a'$. Therefore, $u(a') = h(a)$. Then what we have to show is that $h(a) = \sigma(h(a1), \ldots, h(an))$; but this follows because h is a homomorphism.

Finally, we show that (3) implies (1): since $q: A \to A/Q_f$ is surjective and we have proved that (2) implies (3), we know that q satisfies (3), with the equation $u \circ q = h$ rather than $u \circ f = h$. Hence, taking $h = f$ in (3), we get a unique Σ-homomorphism $u: A/Q_f \to A'$ such that $u \circ q = f$. We now need only show that this is an isomorphism. Using (3a) and (3b) for f, with now $h = q$, we also obtain a unique Σ-homomorphism $u': A' \to A/Q_f$ such that $u' \circ f = q$. We now use (3) for q with $h = q$ also: $q = u' \circ f = (u' \circ u) \circ q$, and also $1_{A/Q_f} \circ q = q$; therefore (3b) gives $u' \circ u = 1_{A/Q_f}$. Similarly, we use (3) for f with $h = f$; then the unique homomorphism is surely $1_{A'}$; but also, $(u \circ u') \circ f = u \circ q = f$, so therefore $u \circ u' = 1_{A'}$. Thus u is an isomorphism. □

For example, let S, Σ, and A be as in the examples above, let Q be $Q8$ and let $B = A/Q2$ with f the natural quotient Σ-homomorphism $A \to A/Q8$. Then u sends (n modulo 8) to (n modulo 2) and preserves oddness.

Define the *image* $f(A)$ of A under $f: A \to B$ to have carriers $f(A)_s = f_s(A_s)$ and operations $\sigma(f_{s1}(a1), \ldots, f_{sn}(an)) = f_s(\sigma(a1, \ldots, an))$ for $\sigma \in \Sigma_{w,s}$ with $w = s1 \ldots sn$, and for $\sigma \in \Sigma_{\lambda,s}$ define $\sigma_{f(A)} = f(\sigma_A)$. Then we can apply Proposition 6 to show that $f(A)$ is Σ-isomorphic to A/Q_f.

4 Abstract data types

This section applies the concepts of Section 3 to data abstraction. Separate subsections consider the case where no equations are needed, and then the use of equations in defining ADTs. Other subsections discuss equational deduction and give several equivalent characterizations of initiality, including a generalized Peano characterization.

We set the stage for what follows by defining the basic concept of an abstract data type. We have already said that a *data representation* is a Σ-algebra. Now notice that the relation of isomorphism is an equivalence relation on the class of all Σ-algebras: any algebra is isomorphic to itself; if

A is isomorphic to B, then B is isomorphic to A; and if A is isomorphic to B and B is isomorphic to C, then A is isomorphic to C. An *isomorphism class* is an equivalence class of Σ-algebras under the equivalence relation of isomorphism, that is, it is a maximal class of Σ-algebras, each of which is isomorphic to all the others.

Definition 7. An *abstract data type* is an isomorphism class of Σ-algebras, for some signature Σ, called the signature of the abstract data type. □

This definition does not address the issue of computability; however, we will see later that algebraic methods can also be used for this.

Initiality provides a particularly elegant way of defining abstract data types. Let us say that a class $\mathscr{C}$ of Σ-algebras is *closed under isomorphism* if for each A member of $\mathscr{C}$, if B is Σ-isomorphic to A, then B is in $\mathscr{C}$. The following is a direct corollary of Proposition 2:

Proposition 8. Let $\mathscr{C}$ be a class of Σ-algebras that is closed under isomorphism and has an initial algebra. Then the class of all algebras that are initial in $\mathscr{C}$ is an isomorphism class. □

Thus, to define an abstract data type, all we have to do is give a class $\mathscr{C}$ of Σ-algebras that is closed under isomorphism and has an initial algebra; then its class of initial algebras will be an abstract data type. Two following subsections consider, respectively, the case where $\mathscr{C}$ is the class of *all* Σ-algebras, and the class of all Σ-algebras satisfying a set E of Σ-equations.

4.1 Data types without equations

Perhaps the most important and familiar abstract data type is the non-negative integers; let us denote this data type N. It can be very simply characterized as a standard algebra (i.e., no junk and no confusion, as in Section 2.3) with signature having only one sort, namely **nat**, one constant 0 of sort **nat**, with one unary function symbol inc: **nat** → **nat**, and with no equations. (Of course inc(n) represents the 'increment' of n, that is, $n + 1$.) Speaking informally, the 'no confusion' condition says that each distinct term inc(inc(...(0)...)) denotes a different number; and the 'no junk' condition says that all numbers are defined by such terms.

The 'initiality' property characterizes the natural numbers more simply but more abstractly by saying that there is one and only one homomorphism from N to *any* other algebra with the same signature. The natural

numbers were first characterized in this way by Lawvere [60]; a proof of equivalence with the usual Peano axioms can be found in MacLane & Birkhoff [64], pages 67–70. This is generalized in Section 4.4 below.

Natural numbers can be denoted in many different ways; each different data representation gives a different algebra, but they are all initial and all isomorphic; in fact, there is a unique isomorphism from any one to any other. The isomorphisms simply give the translations among these representations.

All this generalizes. For any signature Σ, there are many initial Σ-algebras, with the property that there is one and only one Σ-homomorphism from it to any other Σ-algebra; but, any two are Σ-isomorphic, and thus are abstractly the same. We mention two familiar data representations that give initial algebras. In the first, the carrier of sort s consists of all the well-formed Σ-terms of sort s. For the natural number signature, these are just the expressions

$$0, \text{inc}(0), \text{inc}(\text{inc}(0)), \text{inc}(\text{inc}(\text{inc}(0))), \ldots.$$

Such an algebra is called a *term algebra* (or sometimes a *word algebra*) because it consists of all the Σ-terms (or words).†

In the second representation, the carrier of sort s consists of all the well-formed Σ-trees with root of sort s; these can be seen as the parse trees of a grammar $G(\Sigma)$ associated to Σ. For example, a Σ-tree for Σ the signature for vector spaces is given in Figure 14.2; the corresponding Σ-term is $0 + (a \bullet 0)$.

We now make this precise, beginning with an inductive definition of the sets $T_{\Sigma,s}$ of all Σ-*terms*‡ of sort s, for a given S-sorted signature Σ:

(1) $\Sigma_{\lambda,s} \subseteq T_{\Sigma,s}$ for each s in S; and

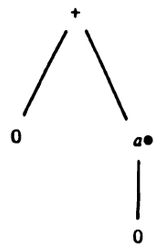


Fig. 14.2. A Σ-tree

† More exactly, these might be called Σ-*ground terms*, to distinguish them from terms that may contain variable symbols; in computer science, terms either with or without variables are called *expressions*.

‡ For T_Σ to satisfy the initiality property, terms should be unambiguous; a sufficient condition for this is to require $\Sigma_{w,s} \cap \Sigma_{w',s'} = \emptyset$ whenever length (w) = length (w') and $(w, s) \neq (w', s')$. This can always be ensured replacing Σ by Σ^e, where $\Sigma^e_{w,s} = \Sigma_{w,s} \times \{(w, s)\}$, and we will assume throughout that terms are always built from such a disambiguated signature whenever Σ is ambiguous.

(2) $\sigma(t1, \ldots, tn)$ is in $T_{\Sigma,s}$ for each σ in $\Sigma_{s1 \ldots sn,s}$ and each ti in $T_{\Sigma,si}$ for $i = 1, \ldots, n$.

We next show that the S-indexed family $T_\Sigma = \langle T_{\Sigma,s} \mid s \text{ in } S \rangle$ is a Σ-algebra, by defining the interpretation $\alpha(\sigma)$ of σ in $\Sigma_{\lambda,s}$ to be the symbol σ in T_Σ (it is in T_Σ by (1)); and defining the interpretation $\alpha(\sigma)$ of σ in $\Sigma_{u,s}$ for $u \neq \lambda$, to send $(t1, \ldots, tn)$ in $(T_\Sigma)^u$ to the term $\sigma(t1, \ldots, tn)$ in $T_{\Sigma,s}$ (it is in T_Σ by (2)).

Two different elements of T_Σ *never* represent the same abstract data item; this 'absolutely no confusion' condition is defined precisely a little later. (These algebras are sometimes called 'absolutely free on zero generators'.) The following very basic result just says that T_Σ is an initial Σ-algebra.

Theorem 9. T_Σ is initial in the class of all Σ-algebras, i.e., for each Σ-algebra A, there is one and only one Σ-homomorphism $T_\Sigma \to A$.

Proof. First notice that T_Σ is by definition a countable union

$$T_\Sigma = \bigcup_n T_\Sigma^{[n]}$$

of S-sorted subsets

$$T_\Sigma^{[0]} = \langle \Sigma_{\lambda,s} \mid s \in S \rangle,$$

$T_\Sigma^{[n+1]} = T_\Sigma^{[n]} \cup \langle \{\sigma(t1, \ldots, tn) \mid \sigma \in \Sigma_{s1 \ldots sn,s} \text{ and } ti \in (T_\Sigma^{[n]})_{si} \text{ for } i = 1, \ldots, n \text{ and } s1, \ldots, sn \in S^*\} \mid s \in S \rangle$. The proof is by induction on n.

Uniqueness: Suppose that h, h': $T_\Sigma \to A$ are two homomorphisms. Then they coincide on $T_\Sigma^{[0]}$ because they preserve the constants:

(i) $h(\sigma) = \sigma = h'(\sigma)$,

and assuming that they coincide on $T_\Sigma^{[n]}$ they coincide on $T_\Sigma^{[n+1]}$, because the homomorphism property and the induction hypothesis give

$$\text{(ii)} \quad h(\sigma(t1, \ldots, tn)) = \sigma(h(t1), \ldots, h(tn)) = \sigma(h'(t1), \ldots, h'(tn)) = h'(\sigma(t1, \ldots, tn)).$$

Thus, they coincide on all of T_Σ as desired.

Existence: Again by induction, we can define h on $T_\Sigma^{[0]}$ by (i), and on $T_\Sigma^{[n+1]}$ by (ii), assuming h already defined on $T_\Sigma^{[n]}$. Thus h is defined on all of T_Σ. □

We now introduce additional basic concepts. A data representation A has *absolutely no confusion* if and only if the unique Σ-homomorphism h: $T_\Sigma \to A$ is injective (i.e., each h_s is injective), and a data representation A

has *no junk* iff h: $T_\Sigma \to A$ is surjective (i.e., each h_s is surjective). A Σ-algebra having no junk is sometimes called *reachable*, *prime*, or *minimal*. Proposition 8 says that for a given Σ, the initial Σ-algebras are an abstract data type; moreover, because an isomorphism is surjective, this ADT (i.e., each member of it) has no junk, and because an isomorphism is injective, it has absolutely no confusion. In fact, these properties characterize this ADT.

Proposition 10. A Σ-algebra A is isomorphic to T_Σ if and only if it has no junk and absolutely no confusion.

Proof. The unique Σ-homomorphism h: $T_\Sigma \to A$ is bijective if and only if it is surjective and injective. □

It may now be worth emphasizing certain points:

1. Notice that we have not *defined* the abstract data type for a given signature (i.e., syntax) Σ to *be* T_Σ; rather, we have let Σ plus the property of initiality define the ADT as an abstract algebra, that is, as an isomorphism class of algebras. For example, there will be one data representation that uses Σ-trees, and another that uses Σ-terms. Approaches that work in terms of one particular model are sometimes called 'abstract model' approaches. However, we do not believe that the word 'abstract' is really appropriate for such approaches (the term 'constructive' is used by Cartwright [19]). Even if one does prefer such an approach, many-sorted general algebra is still a powerful and relevant tool, because such a model actually *is* a many-sorted algebra! Of course, one *can* define an ADT by giving a particular data representation (as a representative of the isomorphism class); the point is then that the class (i.e., the ADT) does not depend upon the choice of representative.
2. We have defined not just the data items of an ADT, but also a complete set of constructors for them; in fact, these constructors define the data items 'abstractly', that is, uniquely up to change of representation.
3. We can speak of 'the' initial algebra for a given signature, because any two are isomorphic, and because we really want to talk about the ADT, that is, about the whole class of isomorphic algebras.
4. Despite this interest in abstraction, it is often necessary to *name* elements of an ADT. The most convenient way to do this is often with terms, i.e., with elements of the term algebra T_Σ. If A is a Σ-algebra, if h: $T_\Sigma \to A$ is the unique Σ-homomorphism, and t is a Σ-term, then $h(t)$ is the element of A *named by* t; this idea was used above to define 'absolutely no confusion' and 'no junk'. It may be useful to think of t as a simple ('straight line') program, of A as a machine that can execute each 'instruction' in Σ, and of $h(t)$ as the result of running t on A.

The term 'finite constructability' is used by Cartwright [19] for the condition of no junk with a finite signature, and he uses the term 'unique constructability' for our 'absolutely no confusion' condition. In addition, Cartwright [19] imposes an 'explicit definability' condition that we will see in Section 6 is unnecessary if the functions involved are computable. Thus, we have here the strange case of an author who not only uses (something exactly equivalent to a special case of) the initial algebra approach to abstract data types without knowing it, but who actually argues in very strong terms *against* using an algebraic approach to data types at all!

4.2 Data types with equations

One might think that for every abstract data type A, there is some finite signature Σ (perhaps contained in the signature of A) such that the abstract data items in A form a Σ-algebra absolutely without confusion. This would mean that the abstract data items are in one-to-one correspondence with Σ-terms (or Σ-trees). Unfortunately, this cannot always be done; some data abstractions are inherently confused. Such data abstractions require the use of equations and of a 'no confusion' condition that is more general than the 'absolutely no confusion' condition. One example of such a data abstraction is that of all the finite SETs of integers with the functions of union, singleton, and epsilon (i.e., 'element of'). The trouble is that union obeys commutative, associative, and idempotent laws.

There are also many cases where one wants to add some auxiliary functions to a given data abstraction, for example, an emptiness test to SET. These auxiliary functions might be defined by some equations in terms of the previously given functions. This subsection generalizes the preceding subsection to permit equations. (However, the 'no confusion' condition is deferred to Section 4.4.)

Fix an S-sorted signature Σ. Now given an S-sorted set X disjoint from Σ, let us think of the elements of X_s (for s in S) as *variable symbols* of sort s, and let us form a new S-sorted signature $\Sigma(X)$ by defining $\Sigma(X)_{\lambda,s} =$

$\Sigma_{\lambda,s} \cup X_s$ and $\Sigma(X)_{w,s}$ for $w \neq \lambda$. We can now form the $\Sigma(X)$-algebra $T_{\Sigma(X)}$ and moreover, we can regard it as a Σ-algebra, denoted $T_\Sigma(X)$, by simply 'forgetting' the variable symbols.† We now define a Σ-*equation* to be a triple $\langle X, t1, t2\rangle$, where X is a finite S-indexed set and $t1, t2$ are in $T_{\Sigma(X),s}$. Given a Σ-algebra A, let us now define an *assignment* from X to A to be a mapping $f\colon X \to A$. Notice that a Σ-algebra A together with an assignment from X to A determines a $\Sigma(X)$ structure on A (just use the assignment to extend the interpretation function of A). Then there is a unique $\Sigma(X)$-homomorphism from $T_{\Sigma(X)}$ to A, i.e., a unique Σ-homomorphism $T_\Sigma(X) \to A$ extending f; let us denote it $f^\#$. We now say that a Σ-algebra A *satisfies* the Σ-equation $\langle X, t1, t2\rangle$ iff for every assignment $f\colon X \to A$, we have that $f^\#(t1) = f^\#(t2)$.

Unfortunately, there is a subtle difficulty with the way that equations are defined in most of the literature (e.g., in Goguen, Thatcher & Wagner [36]). As shown in Section 3.4, to get a deductive system that is sound and complete, it is necessary to *explicitly declare* the variables that are used in each equation. Hence our notation $\langle X, t1, t2\rangle$; we shall also use the perhaps more easily read form

$$(\forall X) \quad t1 = t2,$$

or to make the variables explicit,

$$(\forall x1\colon s1)(\forall x2\colon s2)\ldots(\forall xn\colon sn)\, t1 = t2,$$

where $X_s = \{xi \mid si = s\}$. We shall even allow the familiar notation $t1 = t2$ when the variable declarations are known or obvious.

Given a set E of Σ-equations, let us say that a Σ-algebra A *satisfies* E iff A satisfies each equation in E; in that case, let us call A a (Σ, E)-algebra; we also call (Σ, E) an equational *presentation*. The *variety* of E is the class of all (Σ, E)-algebras. The following generalization of Theorem 9 says that there always are initial (Σ, E)-algebras; it is proved in Section 4.3.3.

Theorem 11. For any signature Σ and set E of Σ-equations, there is an initial (Σ, E)-algebra. □

From the definition of satisfaction it follows easily that the class of all (Σ, E)-algebras is closed under isomorphisms. Thus, the four remarks at the end of Section 4.1 apply as well to the present context where equations are allowed.

† That is, by restricting the interpretation function for $T_{\Sigma(X)}$ from $\Sigma(X)$ to Σ; in general, the algebra resulting from such a restriction of an algebra A to a subsignature Σ is called a Σ-*reduct* and is denoted $A|_\Sigma$.

4.3 Equational deduction

Given a set of equations, the *soundness* of (one-sorted) equational logic asserts that applying a certain set of rules for deducing new equations always yields equations that are satisfied by any algebra satisfying the given equations. Similarly, *completeness* asserts that *every* equation satisfied by all algebras satisfying the given equations can be deduced using these rules. These two properties together imply that, for the class of all algebras satisfying a given set of equations, the *model theoretic* notion of an equation being satisfied by all algebras in the class coincides with the *proof theoretic* notion of the equation being derivable from the given equations by the rules of equational deduction. Such a theorem was first given for the one-sorted case by Birkhoff [15]; see also Tarski [82]. However, neither Higgins [39] nor Birkhoff & Lipson [16] gave rules for equational deduction in their treatments of the many-sorted case. The first completeness theorem for many-sorted equational logic was given by Benabou [2] using a categorical approach, which does not involve explicit rules of deduction. Explicit rules are given in Section 4.3.2 below; soundness and completeness are treated in Section 4.3.3.

In general, the literature on ADTs has simply applied the ordinary rules of one-sorted equational deduction to the many-sorted case. But this is not sound. A first correction of the one-sorted rules by introducing explicit quantifiers yields a system which, although sound, is not complete; further rules are needed for the addition and deletion of quantifiers.

4.3.1 An unsound deduction

The following example demonstrates the unsoundness of using the usual one-sorted rules for many-sorted deduction. Let Σ be the signature with sort set $\{a, b\}$, and with $\Sigma_{\lambda,b} = \{\mathrm{T}, \mathrm{F}\}$, $\Sigma_{b,b} = \{\neg\}$, $\Sigma_{bb,b} = \{\&, +\}$, $\Sigma_{a,b} = \{\mathrm{FOO}\}$, and $\Sigma_{u,v} = \varnothing$ for all other u, v. (Although we intend 'b' to suggest 'Boolean', Σ-algebras need not have as elements of sort b exactly the truth-values T and F; indeed, it may help to think of T and F as two arbitrary symbols that may or may not happen to denote the same element in an algebra.) Finally, let E consist of the following seven equations, where A, B are variables of sorts a, b, respectively

$$\neg \mathrm{T} = \mathrm{F}$$
$$\neg \mathrm{F} = \mathrm{T}$$
$$B + \neg B = \mathrm{T}$$
$$B \,\&\, \neg B = \mathrm{F}$$

$B \,\&\, B = \mathrm{B}$
$B + B = \mathrm{B}$
$\mathrm{FOO}(A) = \neg\mathrm{FOO}(A)$.

Boolean algebra gives the first six equations. The rules of one-sorted equational deduction now give

$$\begin{aligned}\mathrm{T} &= \mathrm{FOO}(A) + \neg\mathrm{FOO}(A) = \mathrm{FOO}(A) + \mathrm{FOO}(A)\\ &= \mathrm{FOO}(A)\\ &= \mathrm{FOO}(A)\,\&\,\mathrm{FOO}(A) = \mathrm{FOO}(A)\,\&\,\neg\mathrm{FOO}(A)\\ &= \mathrm{F}.\end{aligned}$$

If these rules of deduction were sound, then the equation T = F should hold in *every* Σ-algebra satisfying E. But there is a Σ-algebra BAR satisfying E where this is not so: $\mathrm{BAR}_s = \varnothing$; $\mathrm{BAR}_b = \{\mathrm{T}, \mathrm{F}\}$; FOO is the empty function; and all the boolean functions are as expected. Therefore these rules are not sound. This example evolved from one suggested by Gerard Huet, who first pointed out to us the unsoundness of the ordinary rules of deduction in the many-sorted case; it is intended to suggest how unsoundness might arise in practical examples such as parameterized abstract data type definitions.

4.3.2 *The rules of deduction*

The first step toward correcting this situation has already been taken: equations must have all variables explicitly declared with their sorts, yielding what can be thought of as equations with explicit quantifiers. But, if the old one-sorted rules of deduction are modified in this way, the resulting system is *not complete*. Two new rules are needed to add and delete the quantifiers.

Given a signature Σ and a set of Σ-equations, the following are the rules for deriving equations:

(1) *Reflexivity.* Each equation
$(\forall X)t = t$
is derivable.

(2) *Symmetry.* If
$(\forall X)t = t'$
is derivable, then so is
$(\forall X)t' = t$.

(3) *Transitivity.* If the equations
$(\forall X)t = t'$, $(\forall X)t' = t''$
are derivable, then so is
$(\forall X)t = t''$.

(4) *Substitutivity.* If
$(\forall X)t1 = t2$
of sort s is derivable, if $x \in X$ is of sort s', and if
$(\forall Y)u1 = u2$
of sort s' is derivable, then so is
$(\forall Z)v1 = v2$,
where $Z = (X - \{x\}) \cup Y$, $vj = tj(x \leftarrow uj)$ for $j = 1, 2$, and '$tj(x \leftarrow uj)$' denotes the result of substituting uj for x in tj.†

The following two rules complete the system:

(5) *Abstraction.* If
$(\forall X)t = t'$
is derivable, if y is a variable of sort s and y is not in X, then
$(\forall X \cup \{y\})t = t'$
is also derivable. (This rule also applied if $X = \varnothing$, where there are originally no variables, and one is added.)

(6) *Concretion.* Let us say that a sort s is *void* in a signature Σ iff $T_{\Sigma,s} = \varnothing$. Now, if
$(\forall X)t = t'$
is derivable, if $x \in X_s$ does not appear in either t or t', and if s is non-void, then
$(\forall X - \{x\})t = t'$
is also derivable.

4.3.3 *Soundness, completeness, and initiality theorems*

This subsection gives the basic soundness and completeness properties for the rules of equational deduction given in Section 4.3.2. Although the 'ordinary' rules of deduction are not in general sound (Section 4.3.1), it turns out that for many examples of interest they are

† This notion of substitution can be made precise by using the same machinery that was used to define equational satisfaction in Section 4.2. Let t be a Σ-term with variables from X, i.e. $t \in T_\Sigma(X)$. Let $x \in X_s$ and let $u \in T_\Sigma(Z)$, where $Z = (X - \{x\}) \cup Y$. Now define $f\colon X \to T_\Sigma(Z)$ by $f(y) = y$ if $y \neq x$, and $f(x) = u$. Then $f^\#\colon T_\Sigma(X) \to T_\Sigma(Z)$ is the unique Σ-homomorphism extending f, and we define $t(x \leftarrow u) = f^\#(t)$.

both sound and complete; for example, it will suffice for there to be a constant term of each sort. However, there are important examples not having constants of some sorts, such as the theory of partially ordered sets, or many common parameterized ADTs. Theorem 14 below gives a simple necessary and sufficient condition under which the ordinary rules are both sound and complete.

Theorem 12. Soundness. Given a set E of Σ-equations, if an equation is deducible from E using rules (1)–(6), then it is satisfied by every Σ-algebra satisfying E.

The proof, which is a straightforward but tedious check of the soundness of each rule separately, may be found in the appendix. It is interesting to notice that only this result, and not completeness, is needed to prove existence of initial algebras for the equational case, which we restate as

Theorem 11. For any signature Σ and set E of Σ-equations, there is a (Σ, E)-algebra $T_{\Sigma,E}$ that is a quotient of T_Σ such that for any other (Σ, E)-algebra A, there is a unique Σ-homomorphism from $T_{\Sigma,E}$ to A.

Proof. Let Q_E, also denoted Q for short below, be the following Σ-congruence on T_Σ.

tQt' iff $(\forall\varnothing)t = t'$ is derivable from E using rules (1)-(6) of Section 4.3.2.

Rules (1)–(3) give that Q is an equivalence relation. We now show substitutivity. Given σ in $\Sigma_{w,s}$ with $w = s1 \ldots sn$, the equation

$$(\forall X)\sigma(x_1, \ldots, xn) = \sigma(x_1, \ldots, xn)$$

holds by rule (1), with X containing xi of sort si for $i = 1, \ldots, n$. By n applications of rule (4), assuming that $tiQt'i$ for $i = 1, \ldots, n$, we now get

$$(\forall\varnothing)\sigma(t1, \ldots, tn) = \sigma(t'1, \ldots, t'n),$$

and therefore $\sigma(t1, \ldots, tn)Q\sigma(t'1, \ldots, t'n)$ as desired.

Now let $T_{\Sigma,E} = T_\Sigma/Q$. We first show that $T_{\Sigma,E}$ is a (Σ, E)-algebra, i.e., $T_{\Sigma,E}$ satisfies each equation $(\forall X)\ t = t'$ in E. Say X has elements $x1, \ldots, xn$ and consider a map $f: X \to T_{\Sigma,E}$ sending xi to $[ti]$ in $T_{\Sigma,E}$. This map can be factored as $q \circ g$, where $q: T_\Sigma \to T_{\Sigma,E}$ is the quotient and $g: X \to T_\Sigma$ sends xi to ti. By initiality of $T_\Sigma(X)$, we have $f^\# = q \circ g^\#$. Then what we have to show is that $f^\#(t) = f^\#(t')$ or, equivalently, $g^\#(t)Qg^\#(t')$. But (see footnote, Section 4.1) $g^\#(t) = t(x1 \leftarrow t1, \ldots, xn \leftarrow tn)$, and $g^\#(t') = t'(x1 \leftarrow t1, \ldots, xn \leftarrow tn)$. Moreover, the equation

$$(\forall\varnothing)t(x1 \leftarrow t1, \ldots, xn \leftarrow tn) = t'(x1 \leftarrow t1, \ldots, xn \leftarrow tn)$$

is deducible using rule (1) in the form

$$(\forall\varnothing)ti = ti,$$

for $i = 1, \ldots, n$, and n successive applications of rule (4). Thus, $f^\#(t) = f^\#(t')$ as desired.

We now prove that $T_{\Sigma,E}$ is initial. Let A be a (Σ,E)-algebra. By the soundness of rules (1)–(6), A satisfies all equations $(\forall\varnothing)\ t = t'$ deducible from the equations in E using these rules. By the definition of satisfaction, this means that $h(t) = h(t')$, for $(\forall\varnothing)\ t = t'$ any such equation, where $h: T_\Sigma \to A$ is the unique homomorphism. In other words, we have that $Q \subseteq Q_h$, and hence, by Proposition 6, there is a unique Σ-homomorphism $u: T_{\Sigma,E} \to A$ such that $u \circ q = h$, where $q: T_\Sigma \to T_{\Sigma,E}$ is the natural quotient map. All that is now left is to prove uniqueness of u. Any $u': T_{\Sigma,E} \to A$ must satisfy $u' \circ q = h$ since T_Σ is initial for all Σ-algebras; therefore the uniqueness condition of Proposition 6 gives the desired result. □

This construction of the initial (Σ, E)-algebra, $T_{\Sigma,E}$ as a quotient of the term algebra by the congruence Q_E generated by equational deduction from E, is the natural generalization of T_Σ to the case where there are equations. We now state the completeness of our rules of equational deduction; the proof has been exiled to the appendix.

Theorem 13. Completeness. Given a set E of Σ-equations, then every equation satisfied by all the algebras in the variety of E is derivable from E using the rules (1) to (6) above.

We next give necessary and sufficient conditions for the ordinary rules of equational deduction to yield the same derived equations as the rules of Section 4.3.2. Recall that a sort s is *void* in Σ iff $T_{\Sigma,s} = \varnothing$. By an *ordinary equation* of sort s over Σ is meant an expression of the form $t = t'$ where t and t' are both Σ-terms of sort s. Such an equation is *satisfied* in a given Σ-algebra A iff all the equations of the form $(\forall X)\ t = t'$ are satisfied in A, provided that X includes all the variables occurring in t and t'. By the *ordinary rules of equational deduction* we mean the variants of rules (1) to (4) above obtained by eliminating quantifiers. Then, for a given signature Σ, we say that *the soundness and completeness theorems hold in ordinary form* iff for any set E of Σ-equations (with quantifiers), an ordinary equation is satisfied by all algebras satisfying E iff it is derivable using the

ordinary rules of equational deduction. Let $\Sigma\{x:s\}$ denote the new signature constructed from Σ by adding a new constant x of sort s.

Theorem 14. The soundness and completeness theorems hold in ordinary form for a signature Σ iff for all sorts s, s' of Σ, s' is non-void in the signature $\Sigma\{x:s\}$. □

The proof of this result may be found in [32], where necessary and sufficient conditions for the quotient of T_Σ by the Σ-congruence generated by the ordinary rules of deduction to be an initial algebra are also given.

4.4 *Equivalents of initiality*

Some researchers have felt the initiality condition, that 'there exists a unique homomorphism', is too abstract to be of interest (e.g., Cartwright [19]). Yet there are equivalent properties having nothing to do with category theory. This subsection states a number of these, and in particular some generalized Peano axioms.

We now generalize the 'no junk, no confusion' conditions to the case where there may be equations. A (Σ, E)-algebra A has *no confusion relative to* the set E of Σ-equations if and only if the unique Σ-homomorphism $h: T_{\Sigma,E} \to A$ is injective. We now give a corresponding characterization of no junk, and then show that these properties completely characterize the ADT of initial (Σ, E)-algebras.

Proposition 15. A (Σ, E)-algebra A has no junk if and only if the unique Σ-homomorphism $T_{\Sigma,E} \to A$ is surjective. Moreover, A is isomorphic to $T_{\Sigma,E}$ if and only if it has no junk and no confusion relative to E.

Proof. By definition, A has no junk iff $T_\Sigma \to A$ is surjective, and we know from the proof of Theorem 11 that $T_\Sigma \to T_{\Sigma,E}$ is surjective; therefore (by Proposition 3), A has no junk iff $T_{\Sigma,E} \to A$ is surjective. Then $h: T_{\Sigma,E} \to A$ is bijective if and only if it is surjective and injective. □

We now give more concrete equivalents of these conditions, first showing that structural induction [17] is equivalent to the 'no junk' condition, i.e., to reachability.

Proposition 16. The following are equivalent for a Σ-algebra A:

(1) A is reachable, i.e., the unique $h: T_\Sigma \to A$ is surjective;
(2) A has no proper Σ-subalgebras;
(3) *Structural induction.* If $P = \langle P_s \mid s \text{ in } S\rangle$ is an S-sorted subset of A such that
   a. for each constant σ in Σ of sort s, σ is in P_s, and
   b. for each function symbol σ of arity $s1 \ldots sn$ and sort s, if pi is in P_{si} for $i = 1, \ldots, n$, then $\sigma(p1, \ldots, pn)$ is in P_s,
   then $P = A$.

Proof. We first show that (1) implies (2): if A is reachable and P is a proper Σ-subalgebra of A, then $h^{-1}(P)$ is also a proper Σ-subalgebra of T_Σ, which is impossible by Proposition 4.

To show that (2) implies (3), it suffices to note that the conditions (3a) and (3b) say *exactly* that P is a Σ-subalgebra of A.

Finally, to show that (3) implies (1), suppose that h is not surjective. Then its image is a proper subalgebra P of A, i.e., is a subset P of A satisfying (3a) and (3b). □

We now treat the case where there are no equations.

Proposition 17. A is an initial Σ-algebra if and only if it satisfies the following *generalized Peano axioms*:

(1) If σ and σ' are distinct function symbols in Σ of the same sort s, then the images of the functions that they denote on A are disjoint subsets of A_s.
(2) Each σ in Σ denotes an injective function on A.
(3) *Structural induction.* If $P = \langle P_s \mid s \text{ in } S\rangle$ is an S-sorted subset of A such that
   a. for each constant σ in Σ of sort s, σ is in P_s, and
   b. for each function symbol σ of arity $s1 \ldots sn$ and sort s, if pi is in P_{si} for $i = 1, \ldots, n$ then $\sigma(p1, \ldots, pn)$ is in P_s,
   then $P = A$.

Proof. Assume that A is initial. Then by Theorem 9 and Proposition 2, A is isomorphic to T_Σ; so let us assume that A *is* T_Σ. Then axioms (1) and (2) above follow from the construction of T_Σ in Theorem 9. Next, axiom (3) holds by the previous Proposition.

For the converse, assume that A is a Σ-algebra satisfying axioms (1), (2)

and (3). Then the previous Proposition tells us that the unique Σ-homomorphism h: $T_\Sigma \to A$ is surjective. We will be done if we prove that h is also injective, i.e., if we prove that $h(t) = h(t')$ for t, t' in $T_{\Sigma,s}$ implies that $t = t'$. We will prove this by induction on $n = \max\{\text{depth}(t), \text{depth}(t')\}$, where depth($t$) is the depth of t as a tree. For $n = 0$, t and t' are constants in $\Sigma_{\lambda,s}$ and what we want follows from axiom (1). Now suppose $t, t' \in T_{\Sigma,s}$ such that $\max\{\text{depth}(t), \text{depth}(t')\} = n + 1$, and assume (without loss of generality) that $\text{depth}(t) = n + 1$ and $t = \sigma(t1, \ldots, tk)$ for $k > 0$. Then $h(t) = \sigma_A(h(t1), \ldots, h(tk))$, and by axiom (1), t' must be of the form $\sigma(t'1, \ldots, t'k)$ for some $t'1, \ldots, t'k$ in A. Then axiom (2) implies that

$$h(ti) = h(t'i) \quad \text{for } i = 1, \ldots, k.$$

Because each ti and $t'i$ has depth less than or equal to n, the inductive hypothesis gives us that

$$ti = t'i \quad \text{for } i = 1, \ldots, k,$$

and hence that $t = t'$, as desired. □

It follows from the above proof that for reachable algebras, the first two generalized Peano axioms in Proposition 17 are equivalent to 'absolutely no confusion'. This equivalence fails for nonreachable algebras, because the operations may fail to be injective outside the image of T_Σ.

We now further generalize the Peano axioms to include equations.

Theorem 18. Let Σ be an S-sorted signature, let A be a Σ-algebra, and let E be a set of Σ-equations. If t is a Σ-term (i.e., a 'ground' term, containing no variables), let $[t]$ denote the result of evaluating t in A. Then A is initial among all Σ-algebras that satisfy E if and only if

(1) $[t] = [t']$ in A if and only if the equation $(\forall \varnothing)\ t = t'$ can be proved from equations in E using the laws of many-sorted equational deduction given in Section 4.3.2.

(2) *Structural induction.* If P is an S-indexed family of subsets P_s of A such that:
   a. for each constant σ in Σ of sort s, $[\sigma]$ is in P_s and
   b. for each function symbol σ in Σ of arity $s1 \ldots sn$ and sort s, if ai is in P_{si} for $i = 1, \ldots, n$, then $\sigma(a1, \ldots, an)$ is in P_s

   then $P = A$.

Proof. We have only to show that for a reachable Σ-algebra A, axiom (1) is equivalent to 'no confusion', i.e., to injectivity of the unique h: $T_{\Sigma,E} \to A$. If h is injective, then A is isomorphic to $T_{\Sigma,E}$ and $T_{\Sigma,E}$ satisfies axiom (1) by

construction. Conversely, suppose that A is reachable and satisfies axiom (1). Let f: $T_\Sigma \to A$ be the unique Σ-homomorphism. Then A is isomorphic to T_Σ/Q_f and axiom (1) says that $Q_f = Q_E$ the congruence generated by the rules of equational deduction from E. Therefore A is isomorphic to $T_\Sigma/Q_E = T_{\Sigma,E}$ which is initial; thus, h must have been an isomorphism. □

This proof shows that if the second axiom (which is equivalent to 'no junk') is satisfied, then the first axiom is equivalent to 'no confusion'. Thus, when $E = \varnothing$, the first condition above is equivalent to axioms (1) and (2) of Proposition 17. Theorem 18 may not have been formally stated before, but the intuition behind it is part of the folklore.

5 Abstract machines

Recall that *data types* are algebras, whereas *machines* have internal *states* and use techniques and concepts from automaton theory, such as reachability, observability, and minimality. Abstract data types are useful for understanding the type systems of programming languages, especially when they permit user-defined types as in ALGOL 68 [87]. Abstract machines are useful for understanding the specification and implementation of software modules, for example, as in the HDM methodology of [61]. It is a serious error to assume that there is little or no difference between these two enterprises. This error has led, for example, to thinking that the appropriate definition of 'implementation' for software modules is given by the algebraic notion of isomorphism, and has also led to the rather pointless controversy about whether final or initial algebra semantics is ultimately 'the best approach'. For abstract machines, it is their *behavior* that matters. Machines that are different (i.e., non-isomorphic) as data types can still have the same behavior. Thus, a software module can in general be *realized* in many different ways.

Consider for instance the theory of *automata*. It has three sorts, **input**, **state**, and **output**, and operators

__•__: **input, input → input**
λ: **input**
s_0: **state**
next: **input, state → state**
out: **state → output**

plus some obvious equations that make **input** a monoid and *next* a monoid action. An *automaton* is then an algebra on this signature,

satisfying those equations. An automaton becomes a *black box* when we consider the sorts **input** and **output** as the only *visible* sorts. We can then observe the automaton's *behaviour* by feeding it inputs and observing the corresponding outputs. More generally, we are allowed to evaluate in our automaton *all expressions with visible sorts*, such as $out(next(a \bullet c, next(b, s_0)))$, or $\lambda \bullet a \bullet b$ but *not* internal states, i.e., not expressions of sort **state**, such as $next(a \bullet b, s_0)$. Two automata with same input and output sets *have the same behavior*, i.e., are indistinguishable as 'black boxes', if and only if all expressions with visible sort have the same value in both.

All this generalizes for an arbitrary signature Σ and a subset $V \subseteq S$ of sorts declared as *visible sorts*. Let M be a Σ-algebra (we use M to stress that, by specifying which sorts are visible, we are looking at M as a machine) we shall make more precise what we mean by 'evaluating an expression with visible sort' in M. Let M_V denote the V-sorted set $\langle M_v \mid v \in V \rangle$; then the expressions in question are the elements of $T_\Sigma(M_V)_v$, for each visible sort v. There is an evaluation map ε_M that computes the result of evaluating any such expression in M, namely the unique $\Sigma(M_V)$-homomorphism to M (M can be viewed as a $\Sigma(M_V)$-algebra by adding the elements of M_V as constants). We then say that two Σ-algebras, M and M', are *(V-)behaviorally identical*, or that they have the *same (V-)behavior* iff

(i) $M_V = M'_V$, and
(ii) $\varepsilon_M(t) = \varepsilon_{M'}(t)$ for each t in $T_\Sigma(M_V)_v$ with v in V.

For Σ an arbitrary signature, the usual notions of automaton theory generalize to machines, i.e. to Σ-algebras with a given set of visible sorts. For example, every automaton behavior admits an *initial* realization, which is initial among all automata that have that behavior; there is also a *minimal* or *final* realization, having the property of *finality*, dual to initiality, among all reachable realizations of that behavior. The initial realization identifies as few states as possible; the minimal realization identifies internal states as much as possible while retaining the same behavior; thus it uses as few states as possible. The reader may consult [33] for more powerful generalizations of classical automaton theory results (e.g., [28]) to machines; see also [89] and [25]. Here, we construct initial and final realizations of a machine's behavior using definitions which, though not fully general, suffice to present the main results with a minimum of technical machinery.

The initiality and finality theorems for machines use the notion of a *strong* V-homomorphism between two V-behaviorally identical machines. This is exactly a Σ-homomorphism that leaves unchanged all elements of external sort. For M and M' V-behaviorally identical, this can be formulated by saying that $f: M \to M'$ is a *strong V-homomorphism* iff f is a $\Sigma(M_V)$-homomorphism.

Theorem 19. Let Σ be an S-sorted signature, $V \subseteq S$ a set of visible sorts, and M a Σ-algebra. Then there is an algebra $I(M)$ behaviorally identical to M, called the *initial realization* of the behavior of M, such that for any Σ-algebra M' behaviorally identical to M there is a unique strong V-homomorphism $h: I(M) \to M'$.

Proof. $I(M)$ is an S-sorted subset of $T_\Sigma(M_V)$. Specifically, define $I(M)$ to be the (S-sorted) set of V-irreducible terms in $T_\Sigma(M_V)$, where a term t is *V-irreducible* iff any subterm t' of visible sort v is an element of M_v, i.e., iff

$$t = t_1(y \leftarrow t') \text{ for } t' \text{ of sort } v \in V \text{ implies } t' \in M_v.$$

In particular, for each visible sort $v \in V$, we have $I(M)_v = M_v$. The operations of $I(M)$ are defined as follows: the constants for visible sorts are those of M, and for other sorts the constant symbols in the signature. If $\sigma \in \Sigma_{s1 \ldots sn,s}$, and if $t_i \in I(M)_{si}$ for $i = 1, \ldots, n$, then the value of the operation $\sigma(t_1, \ldots, t_n)$ is either the element $\varepsilon_M(\sigma(t_1, \ldots, t_n)) \in M_s$ if the sort s is visible, or else if s is not visible, the term $\sigma(t_1, \ldots, t_n) \in T_\Sigma(M_V)_s$, which is itself V-irreducible since all its subterms $t_1, \ldots, t_n$ are V-irreducible.

From this definition it is clear that M and $I(M)$ are behaviorally identical. Moreover, $I(M) = I(M')$ for any M' (V-)behaviorally identical to M, i.e., $I(M)$ does not depend on the representative M, but only on its behavior.

Note also that there is a unique surjective $\Sigma(M_V)$-homomorphism $\varepsilon_{I(M)}: T_\Sigma(M_V) \to I(M)$. Thus the uniqueness part of the theorem is proved, since there is at most one $\Sigma(M_V)$-homomorphism (i.e., a unique strong V-homomorphism) between $I(M)$ and any M' behaviorally identical to M. To prove the existence part, for any M' that is (V-)behaviorally identical to M, define the function $h: I(M) \to M'$ by $h(t) = \varepsilon_{M'}(t)$. We will be done if we show that

$$(*) \quad h \circ \varepsilon_{I(M)} = \varepsilon_{M'}$$

since then, by Proposition 6, h is a $\Sigma(M_V)$-homomorphism, i.e., a strong V-homomorphism, as desired. Notice that $\varepsilon_{I(M)}(t) = t$ for each $t \in I(M)$; this follows from structural induction over the operations in $I(M)$. Thus for v a visible sort, (*) follows from M' and $I(M)$ both V-behaviorally identical to M. For s a nonvisible sort, note that any term $t \in T_\Sigma(M_V)_s$ can be written as

$$t = t'(x1 \leftarrow t1, \ldots, xn \leftarrow tn)$$

with $t'(x1 \leftarrow \varepsilon_{I(M)}(t1), \ldots, xn \leftarrow \varepsilon_{I(M)}(tn))$ V-irreducible (just take $t1, \ldots, tn$ the largest subterms with visible sorts occurring in t). This finishes the proof, after establishing (by structural induction and the homomorphic property of $\varepsilon_{M'}$) that for any Σ-algebra M' and t as above, the following equality holds

$$(**) \quad \varepsilon_{M'}(t) = \varepsilon_{M'}(t'(x1 \leftarrow \varepsilon_{M'}(t1), \ldots, xn \leftarrow \varepsilon_{M'}(tn))),$$

since then we get

$$\begin{aligned} h(\varepsilon_{I(M)}(t)) &= h(\varepsilon_{I(M)}(t'(x1 \leftarrow \varepsilon_{I(M)}(t1), \ldots, xn \leftarrow \varepsilon_{I(M)}(tn)))) \\ &\quad \text{(by (**))} \\ &= h(t'(x1 \leftarrow \varepsilon_{I(M)}(t1), \ldots, xn \leftarrow \varepsilon_{I(M)}(tn))) \\ &= \varepsilon_{M'}(t'(x1 \leftarrow \varepsilon_{I(M)}(t1), \ldots, xn \leftarrow \varepsilon_{I(M)}(tn))) \\ &\quad \text{(by definition of } h) \\ &= \varepsilon_{M'}(t) \quad \text{(by (**) and behavioral equivalence).} \end{aligned}$$

□

The final realization theorem restricts attention to reachable machines. A Σ-algebra M is called *V-reachable*, or a V-reachable machine, if the evaluation map $\varepsilon_M: T_\Sigma(M_V) \to M$ is surjective. Intuitively this corresponds to not having internal states that cannot be built up from the constants and the visible values by repeated operations. Note that $I(M)$ above is V-reachable by construction. The construction of the final realization identifies any two internal states that cannot be distinguished as different from the visible sorts, i.e., that are (V-)observably equal.

Theorem 20. For Σ an S-sorted signature, $V \subseteq S$ a set of visible sorts, and M a Σ-algebra, there is a V-reachable algebra $N(M)$ that is V-behaviorally identical to M, called the *final* (or *Nerode*) *realization* of the behavior of M, such that for any V-reachable algebra M' behaviorally identical to M there is a unique strong V-homomorphism $q: M' \to N(M)$.

Proof. Define $N(M)$ to be the quotient of $I(M)$ by the following congruence ner: for visible sorts v, $t \,\mathrm{ner}_v\, t'$ iff $t = t'$; for s not in V, $t \,\mathrm{ner}_s\, t'$ iff for each term $t_1 \in T_\Sigma(M_V \cup \{y\})_v$ with y a variable of sort s, $v \in V$, one has

$$\varepsilon_{I(M)}(t_1(y \leftarrow t)) = \varepsilon_{I(M)}(t_1(y \leftarrow t'))$$

Then ner is a Σ-congruence, and (after the trivial identification of each $t \in M_v$ with the one-element equivalence class $\{t\}$) it also follows from the construction that $N(M)$ is behaviorally identical to $I(M)$, thus, also to M. By initiality of $I(M)$, any strong V-homomorphism $q: M' \to N(M)$, for M' V-reachable and behaviorally identical to M must satisfy $q \circ h = p$, for $h: I(M) \to M'$ and $p: I(M) \to N(M)$ the unique strong V-homomorphisms. $\varepsilon_{M'}$ surjective and $\varepsilon_{M'} = h \circ \varepsilon_{I(M)}$ imply h is surjective.

Thus by Proposition 6, q exists and is unique iff the congruence Q_h associated to h is contained in ner. Let $t, t' \in I(M)_s$ be two terms such that $h(t) = h(t')$ but (t, t') is not in ner_s. Then there is a term $t_1 \in T_\Sigma(M_V \cup \{y\})_v$ for some $v \in V$, such that

$$\varepsilon_{I(M)}(t_1(y \leftarrow t)) \neq \varepsilon_{I(M)}(t_1(y \leftarrow t')) \in M_v.$$

Since $\varepsilon_{M'} = h \circ \varepsilon_{I(M)}$ and h_v is the identity on M_v this inequality becomes

$$\varepsilon_{M'}(t_1(y \leftarrow t)) \neq \varepsilon_{M'}(t_1(y \leftarrow t')) \in M_v,$$

which in turn can be expressed as

$$\delta_{M'}(t_1(y \leftarrow t)) \neq \delta_{M'}(t_1(y \leftarrow t')) \in M_v$$

for $\delta_{M'}: T_\Sigma(M') \to M'$ the unique $\Sigma(M')$-homomorphism, since $\delta_{M'}$ extends $\varepsilon_{M'}$. Using the homomorphic property of $\delta_{M'}$ and reasoning by structural induction on t_1 as in (**) of the previous theorem, this inequality can be expressed as

$$\delta_{M'}(t_1(y \leftarrow \delta_{M'}(t))) \neq \delta_{M'}(t_1(y \leftarrow \delta_{M'}(t'))) \in M_v.$$

This is the contradiction we seek, since

$$\delta_{M'}(t) = \varepsilon_{M'}(t) = h(t) = h(t') = \varepsilon_{M'}(t') = \delta_{M'}(t').$$

□

We finish this section by giving a precise definition of an *abstract machine*. Two data types are 'abstractly the same' iff they are isomorphic. Two machines are 'abstractly the same' iff their *behaviors* are isomorphic, i.e., iff (up to a possible change of representation) any expressions with visible sort give the same result in both. Notice that, both for data types and for machines, 'abstract' means (independent of the representation', but in the case of machines this can happen without the machines being isomorphic algebras; only their behaviors have to be isomorphic. In [33] behaviors are actually algebras and the phrase 'isomorphic behavior' has the usual algebraic sense. In this paper we give an equivalent definition that does not require explicitly defining behaviors as algebras, but captures the intuition of visible expressions giving 'the same' result.

Definition 21. For Σ a signature and V a set of visible sorts, one says that two machines M and M' are *V-behaviorally equivalent*, or that they have *isomorphic behaviors*, iff there is a V-sorted bijection $\alpha: M_V \to M'_V$ such that for each t in $T_\Sigma(M_V)_v$ with v in V one has

$$\alpha(\varepsilon_M(t)) = \varepsilon_{M'}(\alpha^\#(t)),$$

where $\alpha^\#$ is the unique $\Sigma(M_V)$-homomorphism induced by the map $M_V \xrightarrow{\alpha} M'_V \xrightarrow{\eta} T_\Sigma(M'_V)$, with η the inclusion. □

It is easy to check that behavioral equivalence is an equivalence relation. Thus we can now define an *abstract machine* (or *abstract module*) as an equivalence class of machines modulo behavioral equivalence. It is also easy to check that if M and M' are behaviorally equivalent then

(i) Their initial realizations $I(M)$ and $I(M')$ are Σ-isomorphic.
(ii) Their final realizations $N(M)$ and $N(M')$ are Σ-isomorphic.

Indeed, if one defines a *V-homomorphism* $f: M \to M'$ as a Σ-homomorphism such that f_v is bijective for each v in V, Theorems 19 and 20 still hold after changing 'behaviorally identical' by 'behaviorally equivalent', and 'strong V-homomorphism' by 'V-homomorphism'. Note finally that the concept of abstract machine generalizes that of abstract data type since in the case where all the sorts are visible two machines are behaviorally equivalent iff they are Σ-isomorphic, i.e., abstract machines become abstract data types when all sorts are visible.

The most common use of final realizations $N(M)$ is to take as M the initial (Σ, E)-algebra, for E a set of equations, and then to take the final realization of its behavior, called the *final algebra* specified by (Σ, E). We shall denote this algebra by $N_{\Sigma,E}$. This is the idea in [38], later formalized by Wand [89]. Note that $T_{\Sigma,E}$ and $N_{\Sigma,E}$ both specify *the same* abstract machine; note also that $I(T_{\Sigma,E})$ is in general not isomorphic to $T_{\Sigma,E}$ but there is a surjective strong V-homomorphism $I(T_{\Sigma,E}) \to T_{\Sigma,E}$.

6 Initiality and computability

This section is a survey of results from a rather widely scattered literature on computable algebras, initiality, and finality, including work by Malcev, Rabin, and Bergstra and Tucker. We stress the fundamental role played by the categories of: (1) recursive sets and recursive functions; and (2) recursive algebras and recursive homomorphisms. The latter category inherits appropriate versions of basic universal algebra constructions such as quotients and free algebras; this helps in establishing facts about computable algebras. Our exposition also includes an introduction to rewrite rules, and a discussion of equality enrichments and their relation to both computability and 'inductionless induction' theorem proving. One new result is an intuitively appealing characterization of computable algebras using only algebraic concepts; this can be seen as a purely algebraic formulation of a Church-like thesis.

6.1 Recursive sets and recursive functions

We assume that the reader is familiar with the intuitive notion of an (*effectively*) *computable* total (or partial) function on the natural numbers; this is a function for which there is an *algorithm* to compute its values. *Church's thesis* identifies this intuitive notion of a computable function with the precise mathematical notion of a ***recursive function***. Recursive functions can be defined in several equivalent ways, such as lambda definability, Turing machines, and primitive recursion with the μ-operator, using 0, the successor function and the projections, where, for $P(x)$ a predicate on the natural numbers, the μ-notation $\mu x[P(x)]$ stands for 'the smallest x such that $P(x)$'.

Unless otherwise stated, by a *recursive* function f on the set ω of natural numbers we will mean a *total* function $f: \omega \to \omega$ that is recursive. A *recursive set* is a subset $U \subseteq \omega$ such that its characteristic function $\chi_U: \omega \to \omega$ is recursive, i.e., such that there is an algorithm to decide whether $n \in U$. The following is a useful technical tool in studying recursive sets.

Lemma 22. Each nonempty recursive set U can be expressed as the image of a recursive *retract*, i.e., of a recursive function $q: \omega \to \omega$ such that $q \circ q = q$.

Proof. Let n_0 be the smallest element of the nonempty set U, i.e., let $n_0 = \mu z[\chi_U(z) = 1]$. Then $q_U: \omega \to \omega$ as given by the λ-expression

$$\lambda x.\ \text{if } x \in U \text{ then } x \text{ else } n_0,$$

satisfies $q_U(\omega) = U$, and is a retract, i.e., $q_U \circ q_U = q_U$.

Conversely, if q is a (recursive) retract, then its nonempty image $U = q(\omega)$ has a recursive characteristic function given by

$$\lambda n.\ \text{if } n = q(n) \text{ then } 1 \text{ else } 0. \qquad \square$$

A *recursive function* $f: U \to V$ between two recursive sets is a total function from U to V that is equal to the restriction of a recursive function on the natural numbers; i.e., there is a recursive $f^0: \omega \to \omega$ such that the diagram in Figure 14.3 commutes (where the vertical arrows denote set inclusions). Under these conditions, we say that f is the *restriction* of f^0, and that f^0 *extends* f.

Lemma 23. Recursive sets and recursive functions are the objects and arrows, respectively, of a category that we shall denote **REC**.

Proof. Since the identity function 1_ω on the natural numbers is recursive and restricts to the identity function 1_U for each recursive set U, it is clear that the identity axiom is satisfied. To see that the composition $g \circ f$ of two recursive functions f and g is recursive, consider the diagram and notice that the rectangle obtained by 'pasting' the two smaller rectangles also commutes. Now, $g^0 \circ f^0$ is recursive, since it is well-known that the composition of two recursive functions on ω gives another recursive function (this is intuitively obvious, since from an algorithm to compute f^0 and another to compute g^0 we can obtain one to compute $g^0 \circ f^0$.) This shows that $g \circ f$ is recursive. □

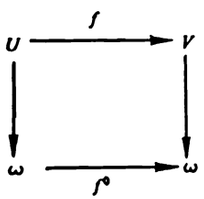
Given a recursive $f^0: \omega \to \omega$ and recursive sets U, V, it may not be *decidable* whether or not f^0 restricts to a function from U to V. However, if V is nonempty, the function $q_V \circ f^0$ will always so restrict, since V is the image of q_V. This provides a fully general method (when V is nonempty) for explicitly presenting a recursive function between recursive sets U, V, namely as a recursive function on ω followed by the retract q_V; for it is straightforward to check that if f^0 restricts to f, then so does $q_V \circ f^0$.

Note also that our definition of a recursive function $f: U \to V$ between two recursive sets captures *all* computable total functions from U to V. For if f is effectively computable and if U is nonempty, then the function $f \circ q_U$ is recursive on ω (by Church's thesis) and extends f (here q_U is understood to have U as its target). Hence, in the sequel we will sometimes define a function with domain U by giving its algorithm, without explicitly mentioning its extension.

Lemma 24. An arrow $f: U \to V$ in **REC** is an isomorphism iff it is bijective.

Proof. Since the arrows of **REC** are functions, it is clear that any isomorphism is a bijection. Conversely, if the recursive function f is bijective, then its inverse function f^{-1} is also recursive, as shown by the expression $\lambda x.\ \mu z[z \in U \text{ and } f(z) = x]$. □

Fig. 14.3. Definition of recursive function



An isomorphism $f: U \to V$ in **REC** will be called a *recursive isomorphism*, and then U and V are said to be *recursively isomorphic*. (Note that this notion of recursive isomorphism is *different* from the standard one in recursion theory.)

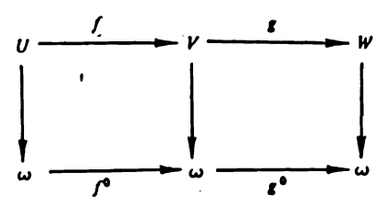
Lemma 25. Each infinite recursive set U is recursively isomorphic to ω; each finite recursive set U, say with $\text{card}(U) = m$, is recursively isomorphic to the set $[m] = \{x \in \omega \mid x < m\}$ of the first m natural numbers.

Proof. If U is empty, then $U = [0]$. If U is a nonempty, the function from U to ω given by the expression $\lambda x.\ \text{card}\{y \in U \mid y < x\}$ defines a recursive isomorphism of U with ω if U is infinite, or with $[m]$ if $\text{card}(U) = m$. □

This proof is not constructive, since we may not be able to decide the cardinality of U from an algorithm to compute its characteristic function (see [80] 5.I, 5.XV).

A *recursive equivalence relation* on a recursive set U is an equivalence relation Q on U such that its characteristic function $\chi_Q: \omega^2 \rightharpoonup \omega$ is recursive, i.e., such that we can *decide* when two elements are Q-equivalent. The equivalence relation Q_f associated to a recursive function $f: U \to V$ is clearly recursive (since $x Q_f y$ iff $f(x) = f(y)$). Conversely, given a recursive equivalence relation Q on U, it is an easy exercise to see that we can define a recursive retract $p_Q: U \to U$ that induces Q, using the expression: $\lambda x.\ \mu z[(z, x) \in Q]$. The retract p_Q picks a canonical representative for each equivalence class modulo Q, namely the smallest element of the class, and so the set $p_Q(U)$ is in bijective correspondence with the set of equivalence classes U/Q. Since $p_A(U)$ is a recursive set and $p_Q: U \to p_Q(U)$ is a recursive surjection, this provides a notion of quotient within the category **REC** (replacing 'equivalence class' by 'canonical representative'). Of course, $p_Q: U \to p_Q(U)$ also has Q as its induced congruence; moreover, the

Fig. 14.4. Associativity of recursive function composition



(recursive) inclusion $j: p_Q(U) \to U$ is a right inverse to p_Q, i.e., $p_Q \circ j = 1_{p_Q(U)}$. More generally, for any surjective recursive function $f: U \to V$ we can find a recursive right inverse g (a 'choice function') such that $f \circ g = 1_V$. One such g is given by the expression $\lambda x.\ \mu z[z \in U \text{ and } f(z) = x]$. There is a lemma of quotients for recursive functions entirely analogous to the one for homomorphisms in Proposition 6.

Proposition 26. Let $f: U \to V$ be recursive function. Then the following are equivalent properties of f:

(1) There is a recursive isomorphism $u: p_{Q_f}(U) \to V$ such that $u \circ p_{Q_f} = f$.
(2) f is surjective.
(3) If $h: U \to A$ is a (not necessarily recursive) function to a set A, then
   a. There exists a function $u: V \to A$ such that $u \circ f = h$ (i.e., the diagram in Figure 14.5 commutes) iff $Q_f \subseteq Q_h$.
   b. If such a function u exists, then it is unique. Moreover, if A and h are recursive, then so is u.

Proof. We first show that (1) implies (2): since p_{Q_f} is surjective and u is an isomorphism, f is also surjective.

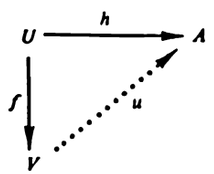
We next show that (2) implies (3b): let g be a recursive right inverse for f. For any u such that $u \circ f = h$ we have

$$u \circ f = u \circ (f \circ g) \circ f = (h \circ g) \circ f.$$

Since f is surjective, this shows that $u = h \circ g$; thus u is unique, and is also recursive if h is.

Assuming (2), we now show (3a). In fact, we will show that $(h \circ g) \circ f = h$ iff $Q_f \subseteq Q_h$. Again letting g be a recursive right inverse for f, $f(g(f(x))) = f(x)$ gives us that $xQ_f g(f(x))$. Then $Q_f \subseteq Q_h$ implies that $h(g(f(x)) = h(x)$ for all x in U. Conversely, if there is an x in U such that $f(x) = f(y)$, but $h(x) \neq h(y)$, then we have $h(g(f(x))) = h(g(f(y)))$ and thus $h \neq (h \circ g) \circ f$.

Fig. 14.5. Universal property of the quotient



Finally, we show that (3) implies (1): since p_{Q_f} and f induce the same recursive equivalence relation and since p_{Q_f} is a retract, the restriction $f': p_{Q_f}(U) \to V$ of f satisfies $f' \circ p_{Q_f} = f$ and also is injective. To see that f' is surjective, let $v: V \to p_{Q_f}(U)$ be the unique recursive function such that $v \circ f = p_{Q_f}$. We then have $f = f' \circ p_{Q_f} = f' \circ v \circ f$, which by (3b) shows $1_V = f' \circ v$; thus $f': p_{Q_f}(U) \to V$ is surjective. □

A slight variant of the above concept of a recursive function between two recursive sets is the concept of a recursive function of several variables: we say that $f: U_1 \times \cdots \times U_k \to V$ is a *recursive function of k variables* (where $U_1, \ldots, U_k, V$ are recursive sets) if f is the restriction of a (total) recursive function $f^0: \omega^k \to \omega$. The same remarks made about recursive functions of one variable apply now, *mutatis mutandis*, to functions of several variables, and show that they capture the concept of 'effectively computable total function of several variables' between recursive sets. Such functions of several variables are used for algebraic operations in the following subsection on recursive algebras.

6.2 Recursive algebras

This subsection introduces the category of recursive algebras (their carriers are recursive sets and their operations are recursive functions) and recursive homomorphisms. The reason for being interested in recursive algebras will be seen better in the next subsection on computable algebras, which shows that several natural definitions of 'computable' for general algebras are equivalent to being isomorphic to a recursive algebra. Here we will see that the category of recursive algebras has initial algebras and, more generally, that any recursive set generates a free recursive algebra. We also look at quotients and congruences of recursive algebras.

Unless otherwise stated, in this and the following subsections, all signatures are assumed finite, i.e., they have a finite number of sorts and a finite number of operators and constants.

Definition 27. A Σ-algebra U is *recursive* if its carrier sets U_s are all recursive sets and its operations are all recursive functions of the appropriate number of variables. A *recursive Σ-homomorphism* $f: U \to V$ between two recursive algebras is a homomorphism such that $f_s: U_s \to V_s$ is recursive for each sort s. □

Recursive algebras and recursive homomorphisms form a category $\mathbf{RALG}_\Sigma$. This follows immediately from the fact that both REC and the category $\mathbf{ALG}_\Sigma$ of ordinary Σ-algebras and their Σ-homomorphisms are themselves categories.

A *recursive* Σ*-congruence* on a recursive algebra U is a congruence Q on U such that Q_s is a recursive equivalence relation for each sort s. The congruence Q_f associated to a recursive homomorphism $f: U \to V$ is clearly recursive. For Q a recursive congruence on U, let us define $q_Q(U)$ as the algebra with carrier $q_{Q_s}(U_s)$ for each sort s, with operations defined by

$$\sigma(n_1, \ldots, n_k) = q_{Q_s}(\sigma(n_1, \ldots, n_k)),$$

and with constants the images of those in U under the maps p_{Q_s}. Then there is a recursive homomorphism $p_Q = \langle p_{Q_s} \rangle: U \to p_Q(U)$ that satisfies the expected property of a quotient.

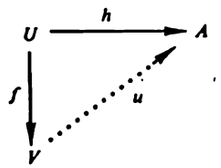
Proposition 28. Let $f: U \to V$ be a recursive Σ-homomorphism. Then the following are equivalent properties of f:

(1) There is a recursive Σ-isomorphism $u: p_{Q_f}(U) \to V$ such that $u \circ p_{Q_f} = f$.
(2) f is surjective.
(3) If $h: U \to A$ is a homomorphism to a (not necessarily recursive) Σ-algebra A, then
   a. there exists a homomorphism $u: V \to A$ such that $u \circ f = h$ (i.e., the diagram in Figure 14.6 commutes) iff $Q_f \subseteq Q_h$.
   b. If such a function u exists, then it is unique. Besides, if A and h are recursive, then so is u.

Proof. Put together Proposition 6 and Proposition 26. □

Recall that if X is an S-sorted set, then $\Sigma(X)$ denotes the signature obtained by adjoining the elements of X as constants to the signature Σ, and $T_\Sigma(X)$ denotes the corresponding initial algebra, also called the *free* Σ*-algebra* on X. If A is a Σ-algebra and if X is an S-sorted set contained in A,

Fig. 14.6. Universal property of the quotient



then the image of $T_\Sigma(X)$ by the unique homomorphism to A (considered as a $\Sigma(X)$-algebra in the obvious way) is called the (Σ-)subalgebra of A *generated* by X. The theorem that follows uses this simple result:

Lemma 29. Let $Y \subseteq X$ be an inclusion of S-sorted sets. Then the Σ-subalgebra of $T_\Sigma(X)$ generated by Y is an initial $\Sigma(Y)$-algebra.

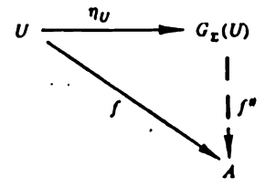
Proof. Since all initial algebras are isomorphic up to a unique isomorphism, we can assume that $T_\Sigma(X)$ and $T_\Sigma(Y)$ are algebras of *terms*. By the construction of term algebras, we then have an inclusion $T_\Sigma(Y) \subseteq T_\Sigma(X)$, which is a $\Sigma(Y)$-homomorphism; thus, its image is initial, since it is $T_\Sigma(Y)$ itself. □

We can now prove the main result of this section, namely that the category $\mathbf{RALG}_\Sigma$ has initial algebras, and more generally, has initial $\Sigma(U)$-algebras for each recursive S-sorted set U; i.e., there are recursive free Σ-algebras. This result is implicit in [67], Theorem 4.1.1 (that paper, together with [78], inaugurated the systematic study of computable universal algebras). Note that the theorem below states the initiality property (expressed in terms of 'universal arrows') not only for the category $\mathbf{RALG}_\Sigma$ but also for the category $\mathbf{ALG}_\Sigma$.

Theorem 30. For each recursive S-sorted set U there is a recursive Σ-algebra $G_\Sigma(U)$ (G for Gödel!) and a recursive S-sorted function $\eta_U: U \to G_\Sigma(U)$ such that for each Σ-algebra A and S-sorted function $f: U \to A$ there is a unique Σ-homomorphism $f^\#: G_\Sigma(U) \to A$ such that the diagram in Figure 14.7 commutes. If A and f are recursive, then so is $f^\#$.

Proof. Let ω denote the S-sorted set with $\omega_s = \omega$ for each sort s. We will define a recursive Σ-algebra structure on ω in such a way that the initial algebras we are looking for will appear as subalgebras of ω. The algebraic

Fig. 14.7. Universal property of a free algebra



structure on ω is defined by means of an assignment of prime numbers to the constants and operation symbols of the signature Σ: each constant σ is assigned a prime number $p(\sigma)$; and each operation symbol of arity $s_1 \dots s_k$ is assigned a sequence $p_1(\sigma) \dots p_k(\sigma)$ of prime numbers. We assume that all the prime numbers are distinct, i.e., that no operators (or constants) have any primes in common, and that the primes in each sequence are distinct. We also assume that the prime 2 does not occur among them. The recursive Σ-algebra structure of ω is defined as follows: the constant σ is the number $p(\sigma)$; if the operation symbol σ has arity $s_1 \dots s_k$, then the corresponding operation is the primitive recursive function

$$\lambda n1, \dots, nk.\ p_1(\sigma)^{n1} \dots p_k(\sigma)^{nk}.$$

Now consider the primitive recursive function $\eta: \omega \to \omega$ defined by $\lambda n.\ 2^n$. This gives an S-sorted function $\eta: \omega \to \omega$ having all components equal to η.

We claim that the subalgebra of ω generated by $\eta(\omega)$, which we denote by $G_\Sigma(\omega)$, is an initial $\Sigma(\omega)$-algebra. This follows easily by checking the Peano axioms of Proposition 17 for $G_\Sigma(\omega)$: The axiom (3) (structural induction) is clearly satisfied since, by definition, this algebra has been obtained as the image of an initial $\Sigma(\omega)$-algebra. Axiom (2) (injectivity of the operations) is satisfied by the algebra ω, by the unique factorization theorem of arithmetic; thus, the operations are *a fortiori* injective when restricted to a subalgebra. To check axiom (1), notice that 0 does not belong to any of the sorts of $G_\Sigma(\omega)$, since it is not in $\eta(\omega)$ and all the operations return values different from 0. Since the prime sequences of each operation symbol are distinct, again by the prime factorization theorem, their images cannot have any value in common. The only exception would be

$$1 = p_1{}^0 \dots p_k{}^0,$$

which has already been ruled out.

Since every recursive set is a subset of ω, every S-sorted recursive set is similarly an S-sorted subset of ω. Since we have shown that $G_\Sigma(\omega)$ is an initial $\Sigma(\omega)$-algebra, Lemma 29 gives that $G_\Sigma(U)$, defined as the subalgebra of ω generated by $\eta(U)$, is an initial $\Sigma(U)$-algebra for each recursive S-sorted set U. To finish the proof, we still have to show that

(i) the $G_\Sigma(U)_s$ are recursive sets, and
(ii) the induced homomorphism $f^\#$ is recursive if f is.

Here is the decision algorithm for $G_\Sigma(U)$: for each integer n of sort s, factor n into its prime factors,

$$n = p_1{}^{n1} \dots p_k{}^{nk}.$$

If $k = 1$ and $p_1 = 2$, then n belongs to $G_\Sigma(U)_s$ iff $n1$ belongs to U_s; otherwise n belongs to $G_\Sigma(U)_s$ iff there is an operation σ of sort s and arity $s_1 \dots s_k$ such that (after eventual reordering of the primes) $p_1 = p_1(\sigma), \dots, p_k = p_k(\sigma)$ and each exponent nj belongs to $G_\Sigma(U)_{s_j}$.

The algorithm for $f^\#$ in terms of a recursive $f: U \to A$ is given by

$$f^\#(2^n) = f(n); \text{ and}$$
$$f^\#(p_1(\sigma)^{n1} \times \dots \times p_k(\sigma)^{nk}) = \sigma(f^\#(n_1), \dots, f^\#(n_k)). \qquad \square$$

Note that the primes in the above theorem could have been chosen in an inifinte number of different ways, as long as they satisfy the conditions of being distinct and being different from a fixed prime (it was 2 above). More generally, it is clear that one could define other recursive functions for the operations σ that would still guarantee the Peano axioms for an algebra generated as in the proof. The particular representation chosen does not much matter; what does matter is that *there is a free recursive Σ-algebra for each recursive S-sorted set.*

This suggests the concept of a Gödel numbering for a free algebra, which numbers the elements of the free algebra in such a way that their images form a recursive free algebra isomorphic to the original one. By the initiality of free algebras, it is enough to number the generators, i.e., to give a function $\eta: X \to U$ to the recursive algebra that provides the numbering. We would also like to require that the map η is somehow 'computable', but since X is not a set of numbers but an arbitrary S-sorted collection of countable sets, the best we can do is require that the image $\eta(X)$ is the image of a recursive function, i.e., is a recursively enumerable set.

Definition 31. A set $Y \subseteq \omega$ is *recursively enumerable* if it is either empty or the image of a total recursive function $f: \omega \to \omega$. Similarly, a set $Y \subseteq \omega^k$ is *recursively enumerable* if it is either empty or of the form

$Y = \{(f_1(n), \dots, f_k(n)) \mid n \in \omega\}$ for some $f_1, \dots, f_k$ total recursive functions. $\square$

Definition 32. Let X be a countable S-sorted set (i.e., each component of X is finite or countably infinite). Then a *Gödel presentation* for X is an S-sorted function $\eta: X \to U$ to a recursive Σ-algebra U such that $\eta(X_s)$ is a recursively enumerable set for each sort s in S and, in addition, η is a universal map in the sense that for each S-sorted $f: X \to A$ to an algebra A there is a unique homomorphism $f^\#: U \to A$ such that $f^\# \circ \eta = f$. The induced isomorphism from the term algebra, $T_\Sigma(X) \to U$, is called the *Gödel numbering* presented by η. $\square$

As a corollary of Theorem 30 we have

Lemma 33. If $\eta: X \to U$ and $\delta: X \to V$ are two Gödel presentations, then U and V are recursively isomorphic Σ-algebras.

Proof. U and V will be recursively isomorphic if we show that both are recursively isomorphic to $G_\Sigma(\mathrm{card}(X))$, where $\mathrm{card}(X)$ is the S-sorted recursive set defined by: $\mathrm{card}(X)_s = [n]$ if $\mathrm{card}(X_s) = n$, and $\mathrm{card}(X)_s = \omega$ if X_s is countably infinite. Since $\eta(X)$ and $\delta(X)$ are recursively enumerable S-sorted sets, by combining Proposition 26 and Lemma 25, there are injective S-sorted recursive functions $\eta_1: \mathrm{card}(X) \to U$ and $\delta_1: \mathrm{card}(X) \to V$ with images $\eta(X)$ and $\delta(X)$ respectively. Then Theorem 30 shows that there are bijective recursive Σ homomorphisms from $G_\Sigma(\mathrm{card}(X))$ to both U and V. By Lemma 24 these two homomorphisms are recursive isomorphisms. □

Note that with $X = \varnothing$, Lemma 33 gives

Lemma 34. Any two Gödel numberings of an initial Σ-algebra T_Σ have recursively isomorphic target algebras; in particular, any such algebra is recursively isomorphic to the algebra $G_\Sigma = G_\Sigma(\varnothing)$. □

We leave the proof of the following lemma as an exercise (hint: use the fact that the generators of $G_\Sigma(\mathrm{card}(X))$ form an S-sorted recursive set).

Lemma 35. The image $\eta(X)$ of a Gödel presentation $\eta: X \to U$ is an S-sorted recursive set. □

6.3 *Computable algebras*

This subsection shows the equivalence of three different definitions for the computable algebra notion. Since each definition is fairly natural and general, their equivalence can be seen as supporting a 'Church's thesis' for effectively computable algebras. We also look at computable minimal algebras (which are computable quotients of the initial algebra), showing that any computable algebra can be seen as a computable minimal algebra if hidden functions are allowed. Our presentation is based upon the work of Malcev, Rabin, and Bergstra & Tucker.

Traditional mathematical practice considers an algebra effectively computable iff it has a 'decidable word problem', meaning that it can be presented as a quotient of a free algebra in such a way that one can decide in a finite number of steps whether or not two terms represent the same element of the quotient algebra (this is called the word problem).

Definition 36. The *word problem is decidable* for a Σ-algebra A iff there is a countable S-sorted set X, a Gödel presentation $\eta: X \to U$, and an S-sorted function $f: X \to A$ such that:

(i) The unique homomorphism $f^\#: U \to A$ induced by f is surjective; and

(ii) the congruence $Q_{f^\#}$ on U is recursive.

By Lemma 33, the choice of Gödel presentation is immaterial, so what really does matter is the map $f: X \to A$, called the *generating* map. We then say that the map f *decides* the word problem for A. ... □

Theorem 37. The following are equivalent for a Σ-algebra A:

(i) The word problem for A is decidable.

(ii) There is a recursive Σ-algebra U and a surjective homomorphism $\alpha: U \to A$ (called a *coordinatization* of A) such that Q_α is recursive.

(iii) A is isomorphic to a recursive Σ-algebra.

Proof

(i) ⇒ (ii). Follows directly from the definition of decidable word problem.

(ii) ⇒ (iii). By Proposition 28, the algebra A is isomorphic to the quotient algebra $p_{Q_\alpha}(U)$.

(iii) ⇒ (i). Let U be a recursive algebra with $\beta: U \to A$ an isomorphism. Take as generating map the map β itself, and as Gödel presentation the map $\eta_U: U \to G_\Sigma(U)$ in Theorem 30. The identity map $1_U: U \to U$ induces a unique surjective recursive Σ-homomorphism $\varepsilon_U: G_\Sigma(U) \to U$. Since β is an isomorphism we have $Q_{\beta \cdot \varepsilon_U} = Q_{\varepsilon_U}$ and this is a recursive congruence. This shows that the word problem for A can be decided by the map $\beta: U \to A$. □

We can now define a *computable* Σ-algebra to be an algebra that satisfies any of the equivalent conditions in the theorem above. For a *finitely generated* Σ-algebra – i.e., a Σ-algebra such that there is a *finite* S-sorted set X and a generating map $X \to A$, i.e., a map such that the induced homomorphism $f^\#$ from the initial $\Sigma(X)$-algebra is surjective – condition (i) takes a stronger form:

Lemma 38. Let A be a finitely generated algebra. If the word problem for A is decidable, then it can be decided by *any* generating map $f: X \to A$ (with X finite).

Proof. By the above theorem, A is isomorphic to a recursive algebra, say U with isomorphism $\beta: A \to U$. Let $f: X \to A$ be a generating map (with X finite) and let $\gamma: X \to \text{card}(X)$ be an arbitrary bijection. We will take as our Gödel presentation the map $\eta_{\text{card}(X)} \circ \gamma: X \to G_\Sigma(\text{card}(X))$. The map $\beta \circ f \circ \gamma^{-1}: \text{card}(X) \to U$ is recursive, since for each sort s with $\text{card}(X)_s = [m]$ and $m > 0$, this map is given by the algorithm: λx. if $x = 0$ then $\beta(f(\gamma^{-1}(0)))$ else ... else if $x = m - 1$ then $\beta(f(\gamma^{-1}(m-1)))$ else $\beta(f(\gamma^{-1}(m-1)))$. Thus by Theorem 30, this map induces a recursive homomorphism $(\beta \circ f \circ \gamma^{-1})^\#: G_\Sigma(\text{card}(X)) \to U$ which, by uniqueness, satisfies $(\beta \circ f \circ \gamma^{-1})^\# = \beta \circ (f \circ \gamma^{-1})^\#$. Thus it is surjective, since $(f \circ \gamma^{-1})^\#$ is so by hypothesis. Now since β is an isomorphism, one has $Q_{(f \circ \gamma)^\#} = Q_{(\beta \circ f \circ \gamma)^\#}$ recursive, as desired. □

Minimal algebras, i.e., algebras such that their unique homomorphism from the initial algebra is surjective, are a particular kind of finitely generated algebra. As a corollary of the lemma just proved, and recalling Lemma 34, we obtain:

Lemma 39. A minimal Σ-algebra A is computable iff Q_{h_A} is a computable congruence on G_Σ where $h_A: G_\Sigma \to A$ is the unique homomorphism. The same holds after replacing G_Σ by any other Gödel numbering of the initial Σ-algebra. □

This subsection concludes by showing that with hidden functions one can reduce the study of computable algebras to that of minimal computable algebras. This uses the following notion:

Definition 40. Given a signature Σ, another signature Σ' (perhaps with more sorts) is called an *enrichment* of Σ if $\Sigma_{w,s} \subseteq \Sigma_{w,s}$ for all w in S^* and s in S; this may be written $\Sigma \subseteq \Sigma'$. The enrichment is called *finite* if each $\Sigma'_{w,s} - \Sigma_{w,s}$ is finite. For Σ' an enrichment of Σ, a Σ-algebra A is called the *Σ-reduct* of a Σ'-algebra A', written $A'|_\Sigma = A$, if the carriers of A and A' coincide, and the operations from the signature Σ are the same for A' as for A; A' is also called an *enrichment of* A. Similarly, a presentation (Σ', E') is an *enrichment* of another presentation (Σ, E) if $\Sigma \subseteq \Sigma'$ and $E \subseteq E'$; the enrichment is called *finite* if both $(\Sigma' - \Sigma)$ and $E' - E$ are finite. An enrichment $\Sigma \subseteq \Sigma'$ or $(\Sigma, E) \subseteq (\Sigma', E')$ is called *without new sorts* if Σ and Σ' have the same sort sets. □

Lemma 41. For any S-sorted computable Σ-algebra A there is a finite enrichment Σ' of Σ without new sorts by at most card(S) constants and card(S) unary function symbols such that there is a minimal computable Σ'-algebra A' which has A as its Σ-reduct.

Proof. To get Σ' from Σ, add a constant *zero* and a unary operation symbol *succ*: $s \to s$ for each sort s in S such that A_s is nonempty. Using Lemma 25, we can show that A is recursively isomorphic to a recursive algebra C with carrier *card*(A). If *card*$(A)_s = \omega$, we make *zero* = 0, and *succ* the successor function; if *card*$(A)_s = [m]$, $m > 0$, we make *zero* = 0, and *succ* the function λx. if $x < m - 1$ then $x + 1$ else $m - 1$, which is clearly recursive. Each of these constants and operations can then be transported to A via the bijection underlying its Σ-isomorphism with C. Together with the original Σ-operations this gives the desired minimal computable Σ'-algebra structure A' with reduct A. □

6.4 *The power of specification techniques: initial algebra semantics*

Let Σ be a signature and E a collection of Σ-equations. A Σ-algebra A is said to have an *initial algebra specification* by means of the presentation (Σ, E) iff A is an initial (Σ, E)-algebra. The specification is called *finitary* if both Σ and E are finite. As we have already seen in Section 4, this provides a specification method whereby certain abstract data types can be defined, and certain concrete data types can be shown to belong to the class of a so-defined abstract data type. For computer science purposes, a specification method should be considered *adequate* (or *powerful enough*) if all computable algebras can be specified with it. This section will show the adequacy of finitary initial abstract data type specifications with hidden functions, i.e., of finite enrichments without new sorts.† Lemma 41 already shows that every computable algebra has an enrichment that is minimal on the enriched signature. This reduces the

† More generally, one could require specifiability of all *semicomputable* algebras (see Definition 61 in Section 6.6) as in Bergstra & Tucker [83], who show that finitary initial algebra specifications are also adequate to specify the larger class of semicomputable algebras if hidden sorts are allowed.

adequacy problem to the specifiability of minimal computable algebras with hidden functions.

The adequacy question for initial algebra semantics was raised by Majster [65], who gave an example of a computable concrete data type (a traversable stack) for which no finitary initial algebra specification existed without the introduction of hidden functions (i.e., without enriching the signature). Majster [66] also gave an explicit definition for computable concrete data types, and suggested (p. 123) using Kleene's normal form theorem to obtain a positive answer to the adequacy problem for initial algebra semantics with hidden functions. The answer along these lines came from Bergstra & Tucker [13], who undertook a rigorous and beautiful systematization of the computability of abstract data types in a rich series of papers. This section will concentrate on the adequacy problem for initial algebra semantics.

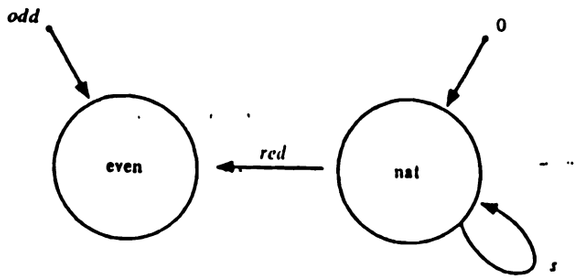
We begin with a simple example (adapted from [84]) of a recursive minimal algebra U which has no finitary initial algebra specification without hidden functions.

The signature of U is shown in Figure 14.8: $U_{nat} = \omega$ and U_{even} is the set of even natural numbers union the number 1. The constant 0 is interpreted as the number 0; the constant *odd* is interpreted as the number 1. The operation *red* is the recursive function λx. if x is-even then x else 1, and the operation s is the successor function.

Lemma 42. No finitary initial algebra specification is possible for U.

Proof. First note that since $U_{nat} = \omega$ and the only operation of sort nat is s, there can be no nontrivial equations of sort nat in any such specification and the only possible equations have to be of sort even; the only terms of that sort are: *odd*, and $red(s^n(0))$ and $red(s^n(x))$ for $n \geq 0$ (using the usual

Fig. 14.8. Signature of the algebra U



conventions that $s^0(x) = x$ and $s^{n+1}(x) = s(s^n(x))$). No equations of the form

$$red(s^n(x)) = red(s^m(x))$$

are possible for m different from n, since we get a contradiction by instantiating x to 0 if n is even, or to 1 if n is odd. Thus we conclude that all equations in E involve ground terms, and are of the form

$$red(s^n(0)) = odd$$

or

$$red(s^n(0)) = red(s^m(0)).$$

To show that no (finitary) initial algebra specification is possible, let E be one, and let $m0$ be the first odd number strictly larger than any of the exponents n, m from the equations in E. Then the equation $s^{m0}(0) = odd$ holds in U, but there is no way to deduce it from the equations in E with the rules of equational deduction. □

The following enrichment permits a (finitary) equational specification of U: add operation symbols *even*: **nat** → **nat**, and *cond*: **nat nat nat** → **nat**, for the recursive functions λx. if x is-even **then 0 else 1**, and $\lambda x, y, z$. **if** $x = 0$ **then** y **else** z, respectively. Thatcher, Wagner & Wright [83] show that this enrichment of U has the following equational initial algebra specification:

$$even(0) = 0$$
$$even(s(0)) = s(0)$$
$$even(s(s(x))) = even(x)$$
$$cond(0, y, z) = y$$
$$cond(s(x), y, z) = z$$
$$red(x) = cond(even(x), red(x), odd).$$

They also show that U has a (finitary) conditional† initial algebra specification with the two conditional equations

$$red(s(0)) = odd$$
$$red(x) = odd \Rightarrow red(s(s(x))) = odd,$$

thus showing that conditional specifications are strictly more powerful than equational specifications if hidden functions are not allowed.

We shall now state the theorem of [13] that gives the definitive answer to the adequacy question for finitary initial algebra specifications. The theorem is stated for minimal algebras but, as proved in the previous subsection, this is not a restriction when hidden functions are allowed. We do not give details of the proof (see the original paper), but just sketch the main lines of their argument.

† See discussion before Definition 64 in Section 6.6.

Theorem 43. Let A be a minimal computable Σ-algebra. Then there is a finite enrichment Σ' of Σ without new sorts and a finite family of Σ'-equations E such that A is isomorphic to the Σ-reduct of the initial (Σ', E)-algebra.

Sketch of the proof. For simplicity, we will reason in the one-sorted case. By Lemma 25, we can replace A either by ω if A is infinite, or by $[m]$ if A is finite. The finite case is straightforward, and reduces in essence to giving a *table* for the operations of A. So we are left with ω, a few numbers corresponding to constants, and a finite collection of (total) recursive functions $f_1, \ldots, f_n$ each having an appropriate number of arguments. The key observation is the following theorem about the graph of a partial recursive function:

Theorem 44 ([68], Thm. 6.1.1). A function is partial recursive iff its graph is recursively enumerable. □

For any recursively enumerable set one can actually find a *primitive recursive* function having that set as its image ([67], Thm. 4.2.1). As a consequence, for each of the functions $f: \omega^k \to \omega$, there are primitive recursive functions $h_1, \ldots, h_k, g: \omega \to \omega$ such that the graph of f is the set:

$$\{(h_1(n), \ldots, h_k(n), g(n)) \mid n \in \omega\}.$$

This suggests specifying the functions f by equations of the form

(i) $f(h_1(x), \ldots, h_k(x)) = g(x)$,

but, of course, we have to specify also the primitive recursive functions $h_1, \ldots, h_k, g$. This is not difficult, since primitive recursive functions are defined equationally in the following way: for g primitive recursive, there is a sequence of (primitive recursive) auxiliary functions $0, s, g_1, \ldots, g_m = g$ such that each g_{i+1} in the sequence is defined equationally in terms of previous functions in the sequence by a pair of equations

(ii) $g_{i+1}(0, x1, \ldots, xq) = g_i(x1, \ldots, xq)$

(iii) $g_{i+1}(s(y), x1, \ldots, xk) = g_j(y, x1, \ldots, xq, g_{i+1}(y, x1, \ldots, xq))$.

Thus, the following enrichment Σ' of the original signature allows everything to be equationally specified: add function symbols for 0, s (successor), and the primitive recursive functions $h_1, \ldots, h_k, g$ associated to each operation f; add also function symbols $g_1, \ldots, g_{m-1}$ for the auxiliary functions of each primitive recursive function g. Let E be the collection of equations of type (i), (ii), (iii), together with an equation

(iv) $\sigma_q = s^n q(0)$

for each constant σ_q of the original signature that was interpreted by the integer n_q. The original algebra is then a reduct of the initial (Σ', E)-algebra. □

See [8] for a different proof of this result in a beautiful theorem that settles in one blow the adequacy question for both initial and final (in the Bergstra and Tucker sense) algebra semantics, and also gives a bound on the number of hidden functions required that is linear in the number of sorts; see Section 6.6. Still another proof of the above theorem follows from Bergstra and Tucker's rewrite rule characterization of computable algebras, discussed in the next subsection.

6.5 *Rewrite rules*

There are close connections between general algebra and rewrite rules. One connection is that equations can be seen as two-way rewriting systems. Another is that rewrite rules provide computationally effective representations for objects that are more abstractly defined by equations plus initiality.

A Σ-equation $(\forall X)\, t = t'$ such that each variable occurring in its left-hand side t also occurs in its right-hand side t', can be used as a *rewrite rule* as follows: a term t_0 can be rewritten to a term t_1 if t_0 contains a subterm that is a substitution instance of the left-hand side t and t_1 is the result of replacing that subterm by the corresponding substitution instance of the right-hand side t'; this is often indicated with the notation $t_0 \to t_1$. Rewriting gives a *unidirectional* version of equational deduction (compare the above with the substitutivity rule). Under mild conditions on a set E of Σ-equations, every term can be rewritten to a unique canonical form. This means that the initial (Σ, E)-algebra is then computable, since we can decide the word problem by rewriting and then comparing canonical forms. A remarkable theorem of [6] shows the converse: any (minimal) computable algebra is the reduct of the initial algebra specified by a finite enrichment without new sorts, whose equations regarded as rewrite rules give canonical forms for the equivalence classes (the minimality restriction can be removed by Lemma 41).

In this way, rewrite rules provide an *operational semantics* for all computable algebras. The *evaluation* of an expression is its canonical form after rewriting, and equality of terms is decided by identity of their canonical forms. This point of view is the basis for the language OBJ [34, 35].

This subsection gives the basic definitions and properties of rewrite rules, and discusses their relationship to initial algebras, ending with the theorem of Bergstra and Tucker mentioned above. For more on rewrite rules, [42] is admirable and complete, and [45] is an excellent survey that is consistent with an algebraic approach.

Given a term $t \in T_\Sigma(X)$, the finite set of variables *occurring* in t, denoted $vars(t)$ is the smallest S-sorted set Z contained in X such that $t \in T_\Sigma(Z)$. That $vars(t)$ is well defined is intuitively obvious, and is formally clear after noticing that: (i) there is always a finite $X' \subseteq X$ such that $t \in T_\Sigma(X')$; and (ii) if t is $T_\Sigma(Y)$ and in $T_\Sigma(Z)$, then t is in $T_\Sigma(Y \cap Z)$. We will say that an equation $(\forall X)\ t = t'$ is *usable* as a rewrite rule if t has all the variables that t' has (i.e., $vars(t') \subseteq vars(t)$) and in addition, the quantified variables include only those occurring in t (i.e., $X = \text{vars}(t)$). If these two conditions are satisfied, we can omit the quantifier without introducing any ambiguity. Usable equations support term rewriting. For example, the equation $x + s(y) = s(x + y)$ can be used to rewrite the term $(s(x) + s(z)) + (y + (s(x) + s(z)))$ to $(s(x) + s(z)) + (y + s(s(x) + z))$ by matching the left-hand side $x + s(y)$ with the second occurrence of the subterm $s(x) + s(z)$. Now the formal definition.

Definition 45. A *matching* of a term t with a subterm of another term $t_0 \in T_\Sigma(Z)$ is a pair (f, v) with f an *assignment* f: $\text{vars}(t) \to T_\Sigma(Z)$ and with $v \in (T_\Sigma(Z \cup \{y\}) - T_\Sigma(Z))$ a term having exactly one occurrence of the variable y (i.e., there is no v' with $\text{vars}(v') = Z \cup \{u, w\}$, $u \neq w$, and $v = v'(u \leftarrow y, w \leftarrow y)$) such that $t_0 = v(y \leftarrow f^\#(t))$.† A set E of usable equations defines a binary relation $\to_X$ on the term algebra $T_\Sigma(X)$, called *one step* (E-)*rewriting*, as follows: for any two terms t_0 and t_1 we have $t_0 \to_X t_1$ iff there is an equation $t = t'$ in E such that t matches a subterm of t_0 by (f, v) (i.e., $t_0 = v(y \leftarrow f^\#(t))$ and also $t_1 = v(y \leftarrow f^\#(t'))$. □

Notice that $t_0 \to_X t_1$ iff $t_0 \to_{vars(t_1)} t_1$. This is because, if the rewriting was obtained by a matching (f, v) of the right-hand side of a usable equation $t = t'$, then the image of the homomorphism $f^\#$ is always contained in $T_\Sigma(vars(t_1))$, and $vars(t_1) \subseteq vars(t_0)$ since $vars(t') \subseteq vars(t)$ by hypothesis. As a consequence, the relation $\to_X$ restricts well to term algebras with fewer variables, i.e.,

$$\to_X |_{T_\Sigma(Y)} = \to_Y$$

whenever $Y \subseteq X$, and we can therefore drop subscripts on $\to$. But unless

† This includes the case in which the term v is the variable y, i.e., the subterm matched is t_0 itself.

otherwise specified, the rest of our discussion will assume the rewriting relation $\to$ is restricted to the initial algebra T_Σ; this restriction involves no loss of generality since, for any X, $T_\Sigma(X)$ is the initial $\Sigma(X)$-algebra, and all the results we discuss specialize to terms with variables by taking $\Sigma(X)$ as the original signature. In this subsection, signatures are not assumed finite.

Let $\overset{*}{\to}$ denote the reflexive-transitive closure of the one step rewriting relation associated to a set of usable equations E; i.e., $t \overset{*}{\to} t'$ iff either $t = t'$ or there is a finite sequence of one-step rewritings beginning with t and ending with t':

$$t \to t_1 \to \ldots t_k \to t'.$$

We call $\overset{*}{\to}$ the *rewriting relation* associated with E. Also we let $\overset{*}{\leftrightarrow}$ denote the smallest equivalence relation containing $\to$. This equivalence relation is easily described in terms of yet another relation $\updownarrow$, defined as follows: $t \updownarrow t'$ iff there is a term t'' such that $t \overset{*}{\to} t''$ and $t' \overset{*}{\to} t''$.

Lemma 46. The equivalence relation $\overset{*}{\leftrightarrow}$ is the transitive closure of the relation $\updownarrow$. In other words, $t \overset{*}{\leftrightarrow} t'$ iff there is a sequence $t \updownarrow t_1 \updownarrow \ldots \updownarrow t_k \updownarrow t'$ as shown in Figure 14.9.

Proof. It is clear that the relation $\updownarrow$ contains $\to$ and is contained in $\overset{*}{\leftrightarrow}$, since the same conditions hold for $\overset{*}{\to}$, and $\overset{*}{\leftrightarrow}$ is symmetric and transitive. Thus, the transitive closure $\updownarrow$ is also contained in $\overset{*}{\leftrightarrow}$, since $\overset{*}{\leftrightarrow}$ is transitive. So, we have only to show that this transitive closure is reflexive and symmetric. But both these follow from $\updownarrow$ being reflexive and symmetric. □

Lemma 47. The relation $\overset{*}{\leftrightarrow}$ is a Σ-congruence on T_Σ.

Proof. We have to show that for each operation $\sigma \in \Sigma$ and for all pairs $t_i \overset{*}{\leftrightarrow} t_i'$ for $1 \leq i \leq n$ (of the appropriate arity), one has $\sigma(t_1, \ldots, t_n) \overset{*}{\leftrightarrow} \sigma(t_1', \ldots, t_n')$. The key observation is that if a term t matches a subterm of another term t', then it also matches the same subterm for any

Fig. 14.9. The equivalence $\overset{*}{\leftrightarrow}$

term t'' that has t' itself as a subterm. As a consequence, if $t_1 \to t_2$ then $t(y \leftarrow t_1) \to t(y \leftarrow t_2)$ for any $t \in T_\Sigma(\{y\})$. Hence the sequence

$$t_1 \updownarrow t_{11} \updownarrow \dots \updownarrow t_{1k} \updownarrow t_1''$$

yields another sequence

$$\sigma(t_1, \dots, t_n) \updownarrow \sigma(t_{11}, \dots, t_n) \updownarrow \dots \updownarrow \sigma(t_1'', t_2, \dots, t_n).$$

Consequently, we get

$$\sigma(t_1, \dots, t_n) \overset{*}{\leftrightarrow} \sigma(t_1'', t_2, \dots, t_n) \overset{*}{\leftrightarrow} \dots \overset{*}{\leftrightarrow} \sigma(t_1'', \dots, t_n''),$$

as desired. □

We can now prove the central property of the rewriting relation [30].

Theorem 48. Let $\to$ be the one-step rewriting associated with a set E of usable equations. Then the algebra $T_\Sigma \overset{*}{\leftrightarrow}$ is the initial (Σ, E)-algebra.

Proof. By Theorem 11, we know that the initial (Σ, E)-algebra is T_Σ/Q_E with tQ_Et' iff $(\forall \varnothing)t = t'$ can be deduced from E by the rules of equational deduction. Whenever $t_1 \to t_2$ holds, there is an equation $(\forall X)\, t_1' = t_2'$ in E, a term $v \in T_\Sigma(\{y\})$, and a map $f: X \to T_\Sigma$ such that $t_i = v(y \leftarrow f^\#(t_i'))$ for $i = 1, 2$. Consequently, $(\forall \varnothing) f^\#(t_1') = f^\#(t_2')$ can be deduced by $\text{card}(\bigcup_s X_s)$ applications of the substitutivity rule, and $t_1 Q_E t_2$ can be obtained by one more application of that rule. This shows that $(\to) \subseteq Q_E$ which by the reflexivity, symmetry and transitivity rules of deduction, shows that $\overset{*}{\leftrightarrow} \subseteq Q_E$. So we will be done if we show that $T_\Sigma/\overset{*}{\leftrightarrow}$ satisfies the equations in E. Let $(\forall X)\, t = t'$ be any such equation. By definition of $\to$ we then have that $f^\#(t) \to f^\#(t')$ for each map $f: X \to T_\Sigma$ and hence the equation holds in $T_\Sigma/\overset{*}{\leftrightarrow}$. □

This shows that for constructing initial algebras, the unidirectional deduction provided by the rewriting relation is as good as the usual equational deduction (see also [88]), but it does not show any computational advantage of rewrite rules. In fact, it cannot do this, since the theorem applies to any set E of (usable) equations, and it is well known [40] that there are (finitary) initial algebra specifications (Σ, E) with undecidable word problems; nothing can be done in those cases to solve the word problem, regardless of the kind of deduction used.

We will soon see that if the rewrite rules satisfy two natural conditions then the word problem is decidable, and can be decided by rewriting. We will also see that this method is fully general: any minimal algebra with a decidable word problem is the reduct of the initial algebra specified by a

finite enrichment without new sorts and a finite set of usable equations whose rewriting relation decides it.

Definition 49. Let E be a set of usable equations and let $\to$ be its one-step rewriting relation. Then a term t_0 is a *normal form* relative to $\to$ if it cannot be further rewritten; i.e., if there is no t_1 such that $t_0 \to t_1$. The relation $\to$ is called *terminating* if there is no infinite sequence of rewritings

$$t_0 \to t_1 \to \cdots \to t_n \to \cdots.$$

Notice that if a system is terminating, then every term rewrites to a (not necessarily unique) normal form (called 'a normal form of t') after a finite number of rewritings. □

An example of non-termination is given by the commutativity law for addition, which yields infinite rewritings like

$$3 + 2 \to 2 + 3 \to \cdots \to 3 + 2 \to 2 + 3 \to \cdots.$$

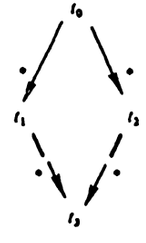
Similarly, an equation of the form $x = \sigma(x)$ gives an infinite rewriting

$$\sigma_0 \to \sigma(\sigma_0) \to \cdots \to \sigma^n(\sigma_0) \to \cdots$$

for a constant σ_0. Intuitively, for $\to$ to be terminating, the 'size' of terms should decrease after rewriting, for some notion of 'size' suited to the problem at hand.

Definition 50. Let E be a set of usable equations and let $\to$ be its corresponding one step rewriting relation. Then $\to$ is called *confluent* (or *Church–Rosser*) if for each term t_0 and each pair of rewritings $t_0 \overset{*}{\to} t_1$ and $t_0 \overset{*}{\to} t_2$ we have that $t_1 \updownarrow t_2$, i.e., that t_1 and t_2 rewrite to a common term t_3. □

Fig. 14.10. The Church–Rosser property



Theorem 51 (after [31]). Let Σ be a finite signature, and let E be a finite set of usable Σ-equations such that the corresponding one step rewriting relation $\to$ is terminating and confluent. Then:

(i) Each term t has a unique normal form, denoted $[\![t]\!]$. For each pair of terms, t, t', one has $t \overset{*}{\leftrightarrow} t'$ iff $[\![t]\!] = [\![t']\!]$.
(ii) The initial (Σ, E)-algebra is computable.
(iii) The (S-sorted) set of normal forms, denoted $\mathrm{Can}_{\Sigma,E}$ with constants $[\![\sigma]\!]$ and operations $\sigma([\![t_1]\!], \ldots, [\![t_n]\!]) = [\![\sigma(t_1, \ldots, t_n)]\!]$ is an initial (Σ, E)-algebra, called the *canonical term algebra* associated to E.

Proof. Both (ii) and (iii) follow from (i), since rewriting provides an algorithm to decide the word problem in the initial (Σ, E)-algebra; by Church's thesis this algorithm corresponds to a recursive congruence on G_Σ and this is our formal definition for decidability of the word problem; thus we get (ii). Again using (i), $\mathrm{Can}_{\Sigma,E}$ is isomorphic by construction to the initial algebra $T_{\Sigma,E}$ formed by the equivalence classes of terms. The isomorphism is the map can: $[t] \to [\![t]\!]$, where $[t]$ is the E-equivalence class of t; this gives (iii).

Now let us prove (i). Since $\to$ is terminating, each term t has at least one normal form t_1; suppose it has a second normal form t_2. By confluence there is a t_3 such that both t_1 and t_2 rewrite to t_3. Since t_1 and t_2 are normal forms, this can only happen if $t_1 = t_3 = t_2$. If $[\![t]\!] = [\![t']\!]$, then $t \overset{*}{\leftrightarrow} t'$. That $t \overset{*}{\leftrightarrow} t'$ implies $[\![t]\!] = [\![t']\!]$ follows by induction on the length of the sequence $t \updownarrow \ldots \updownarrow t'$, using confluence; this is left to the reader. □

Given a set of equations, if we can show that they are terminating and confluent, then by the above theorem, we have solved the word problem for its initial algebra. For recent methods to establish termination of a set of rewrite rules see [76, 22, 52, 23]. We will now discuss methods to establish confluence. The idea is to reduce confluence to a simpler condition of 'local confluence' which is decidable, providing termination holds (this can be relaxed, as explained below).

Definition 52. Let E be a set of equations and let $\to$ be its corresponding one-step rewriting relation. Then $\to$ is *locally confluent* if for each term t_0 and each pair of one-step rewritings $t_0 \to t_1$, $t_0 \to t_2$ we have that $t_1 \updownarrow t_2$. □

The following result is originally due to Newman [73]; a simple proof by 'Noetherian induction' can be found in [42]. Although this result can be stated very generally for an abstract relation, we specialize it to rewriting systems.

Proposition 53. Let $\to$ be a terminating rewriting relation associated to a set E of equations. Then $\to$ is confluent if and only if it is locally confluent. □

The Knuth–Bendix algorithm tests for local confluence of a set of equations, and can also be used to attempt completing a nonconfluent set of equations into an equivalent set of (locally) confluent equations. What follows is an informal introduction to the main ideas and extensions of this algorithm; technical details can be found in the cited references.

Let X be a fixed S-sorted set with an infinite number of variables of each sort, and consider the rewriting relation $\to$ on $T_\Sigma(X)$. Huet [42] has shown that local confluence of the rewriting relation $\to$ is decidable by the Knuth–Bendix algorithm provided that the relation is terminating. The idea is that any pair of one step rewritings either can trivially be shown to rewrite to a common term, or else is a specialized instance of a finite set of 'most general' pairs of one step rewritings, called 'critical pairs', that can be obtained by 'superposing' pairs of equations in E in a 'most general' way. Since the relation is assumed terminating, we can decide local confluence (hence confluence) simply by comparing normal forms for each side of the pair. The Knuth–Bendix algorithm [55] finds all the critical pairs of a set of equations. For example, if E contains the equations

$$\sigma(\tau(x, y), z) = \mu(x, y, z)$$
$$\tau(x, \eta(y)) = \kappa(y, x)$$

then $\langle \sigma(\kappa(y, x)), z);\ \mu(x, \eta(y), z)\rangle$ is a critical pair. Even if a set of equations does not give a confluent rewriting relation, in many common cases it can be replaced semiautomatically by an equivalent one that does, using the Knuth and Bendix algorithm. The idea is to choose a good orientation for the normal forms of each critical pair that does not have a common rewriting, and iterate the algorithm until this does not happen anymore. Unfortunately, this process may not stop.

The Knuth–Bendix algorithm has been extended to handle special cases,

Fig. 14.11. The local confluence property

such as a commutative equation, that give a nonterminating rewriting relation. The idea is to split the equations in E into a set of 'rewriting' rules E_1 and a set of 'equivalence' rules E_2 and then to consider ways of rewriting with E_1 'modulo E_2'. Three basic papers are: [58, 75, 42]. For the present state of the art see [49, 50]. The nonterminating case requires in addition a complete unification algorithm for the 'equivalence' equations E_2 to compute 'complete sets of critical pairs' modulo the equivalence. Such algorithms are known for various special cases such as commutativity [77] and associativity-commutativity [81]. General methods for building such algorithms are studied in [46, 51].

We now conclude this section with the theorem of [6] showing the converse, that any computable (minimal, but we know this is no loss of generality) algebra has an initial algebra specification by a finite enrichment without new sorts of its signature and a finite number of equations that yield a terminating and confluent rewriting system. As before, details of the proof are not given, but the main lines of the argument are sketched.

Theorem 54. Let Σ be a finite signature and A a minimal Σ-algebra. The following are equivalent:

(i) A is computable.

(ii) There is a finite enrichment without new sorts Σ' of Σ and a finite set E of usable equations such that induced rewriting relation $\rightarrow$ is terminating and confluent, and A is isomorphic to the Σ-reduct of the canonical term algebra $\mathrm{Can}_{\Sigma',E}$.

Sketch of the proof. Theorem 51 shows that (ii) $\Rightarrow$ (i).

To see that (i) $\Rightarrow$ (ii), we consider the one-sorted case for simplicity. A can be taken to be ω or $[m]$. The finite case is easy, since for any finite algebra the tables of their operations provide a terminating and confluent rewriting relation. Thus, we need only consider A with carrier ω, a finite collection of numbers for the constants and a finite collection of (total) recursive functions $f_1, \ldots, f_k$ each having the appropriate number of arguments. We use a fundamental result in the theory of recursive functions, Kleene's enumeration theorem:

Theorem 55 ([68], Thm. 6.2.1). Every partial recursive function $f(x_1, \ldots, x_n)$ can be written in the form

$$f(x_1, \ldots, x_n) = \mathit{left}(\mu z[F(x_1, \ldots, x_n, z) = 0]),$$

where *left* is the (primitive recursive) left projection function for the Cantor

diagonal enumeration of ω^2, *cant*: $\omega^2 \rightarrow \omega$. i.e., $\mathit{left}(\mathit{cant}(n, m)) = n$, and F is a primitive recursive function depending on f. □

We can apply this theorem to each operation f: $\omega^n \rightarrow \omega$ of our algebra, and define two auxiliary primitive recursive functions h, g by the equations

$$h(z, x_1, \ldots, x_n) = \mathit{left}(\mu z' \leq z[z' = z \text{ or } F(x_1, \ldots, x_n, z') = 0]),$$

$$g(z, x_1, \ldots, x_n) = \text{if } \exists z' \leq z[F(x_1, \ldots, x_n, z) = 0] \text{ then 0 else 1},$$

and an auxiliary recursive function t defined by the equations

(i) $t(z, x_1, \ldots, x_n, 0) = h(z, x_1, \ldots, x_n)$

(ii) $t(z, x_1, \ldots, x_n, y + 1) = t(z + 1, x_1, \ldots, x_n, g(z + 1, x_1, \ldots, x_n))$.

It is then easy to check that f can be defined by the equation

(iii) $f(x_1, \ldots, x_n) = t(0, x_1, \ldots, x_n, 1)$.

The enriched signature Σ' is obtained by adding the following function symbols: 0; the successor function s; the functions g, h, t for each operation f; and function symbols for each of the primitive recursive functions $g_1, \ldots, g_m, h_1, \ldots, h_m$ needed to define each g and h from 0 and s by primitive recursion for each operation f. The equations in E are as follows:

(1) The equations defining each g, h, and their auxiliary functions g_i and h_j by primitive recursion for each f.

(2) The equations (i)-(ii) for each f.

(3) An equation $\sigma q = s^{nq}(0)$ for each constant σq of the original signature which was interpreted by the number nq.

One must then verify that this specification induces a terminating and confluent rewriting relation that has ω as its canonical term algebra. This is done in two steps, with the following lemmas:

Lemma 56. If t is a terminating term for (Σ', E) (i.e., if there is no infinite sequence of rewritings beginning with t), then t has a unique normal form of the form $s^n(0)$. □

Lemma 57. The relation $\rightarrow$ associated to (Σ', E) is terminating. □

The proof of the last lemma uses induction on the depth of terms and case analysis. Note that confluence follows from termination by the existence of a unique normal form for each term. □

6.6 *The power of specification techniques: final algebra semantics*

To examine the relationship between final algebra semantics and

computability, it is reasonable to restrict attention to machines (i.e., algebras) that have *computable behavior*, since this is clearly necessary for a *computable realization* of that behavior to exist. By 'computable behavior' we mean that the word problem is solvable for the visible sorts. Here is the definition.

Definition 58. For Σ a finite signature and V a subset of sorts, a Σ-algebra M has a *computable V-behavior* iff there is a V-sorted recursive set W and a V-sorted map $f: W \to M$ such that the induced homomorphism $f^\#: G_\Sigma(W) \to M$ is surjective in each component $f^\#_v$ with $v \in V$, and the equivalence relation $(Q_{f^\#})_v$ is recursive for each $v \in V$. □

Since $G_\Sigma \subseteq G_\Sigma(W)$ for any W, if M is a minimal Σ-algebra, then the above definition can be rephrased by saying that M has a computable V-behavior iff $(Q_h)_v$ is a recursive equivalence relation for each $v \in V$, for $h: G_\Sigma \to M$ the unique homomorphism. Note also that this definition is stable under isomorphism: if M has a computable behavior so does any M' isomorphic to M.

The following theorem of Bergstra & Meyer [5] shows that every computable behavior is *effectively realizable* by the initial realization.

Theorem 59. Let M be an algebra with a computable V-behavior, for V a subset of its sorts. Then the initial realization $I(M)$ is a computable algebra.

Proof. If $f: W \to M$ shows that M has a computable behavior, then by Proposition 26 we can find recursive sets U_v and recursive retracts $p_v: G_\Sigma(W) \to U_v$ such that $f^\#_v = g_v \circ p_v$ with $g_v: U_v \to M_v$ bijective for each v in V. Since computability of an algebra is a concept stable under isomorphism, without loss of generality we may assume that the bijection $g: U \to M_V$ is the identity. Recall that $I(M)$ was constructed as the V-sorted subset of $T_\Sigma(U)$ formed by the V-irreducible terms. Notice that there is an algorithm to decide if a term t is irreducible: just examine all proper subterms of t and see if there is one of external sort different from a constant in U. Consequently, $I(M)$ is in bijective correspondence with an S-sorted recursive subset by the unique isomorphism of $G_\Sigma(U)$ with $T_\Sigma(U)$ as $\Sigma(U)$-algebras, since this later isomorphism is given by an algorithm, and we can then use Church's thesis. Using this bijection and again by Church's thesis, each operation of $I(M)$ corresponds to a recursive function for the image, i.e., $I(M)$ is a computable algebra, since, for σ with

nonvisible sort, the algorithm to compute σ is the same as the one to compute σ in $T_\Sigma(U)$, and, for σ with visible sort, the algorithm to compute $\sigma(t1, \ldots, tn)$ is as follows: (i) compute $a1, \ldots, an$, the Gödel numbers of $t1, \ldots, tn$, in $G_\Sigma(U)$; then the result is $p_v(\sigma(a1, \ldots, an))$. □

As a corollary to this theorem, we next show that (V-reachable) algebras with computable behavior are always minimal in a finite enrichment without new sorts of their original signature. Recall that an algebra is V-reachable, for V a set of sorts, iff the evaluation map $\varepsilon_M: T_\Sigma(M_V) \to M$ is surjective; thus any minimal Σ-algebra is V-reachable, but a V-reachable algebra need not at all be minimal.

Lemma 60. For Σ a finite signature, let M be a V-reachable Σ-algebra with a computable V-behavior. Then there is an enrichment without new sorts Σ' of Σ by at most $|V|$ constants and $|V|$ function symbols, and a minimal Σ'-algebra M' with a computable V-behavior (as a Σ'-algebra) such that $M'|_\Sigma = M$.

Proof. By Theorem 59, $I(M)$ is computable; therefore, it is isomorphic to a recursive number algebra U, which we may assume for each sort s has either $U_s = \omega$ (if U_s is infinite), or $U_s = [n]$ for some integer n (if U_s is finite). For each visible sort v such that U_v is nonempty, we can pick $0 \in U_v$ as a constant and a recursive function $s: U_v \to U_v$ that is the ordinary successor function when U_v is infinite, or the truncated successor function,

$$\lambda x.\ \text{if } x < n - 1 \text{ then } x + 1 \text{ else } n - 1,$$

when $U_v = [n]$. This makes U into a recursive Σ'-algebra, U', for the signature obtained by adding a constant 0 and a successor function s to each visible sort v with U_v nonempty. Using the bijection between $I(M)$ and U, we can then make $I(M)$ into a computable Σ'-algebra, $I(M)'$, and, since $I(M)_v = M_v$ for each visible sort V, this also makes M into a Σ'-algebra M'.

(Incidentally, $I(M)'$ is the initial realization of M'.) M' is the algebra we want; since $I(M)'$ is generated by M_V as a Σ-algebra, then $I(M)'$ is a minimal Σ'-algebra, and so is M' for the same reason. The unique Σ'-homomorphism $h: G_{\Sigma'} \to M'$ factors through the quotient $I(M)' \to M'$ and $I(M)'$ is computable, so M' has a computable V-behavior, as desired. □

Before considering the relationship between final algebra semantics and computability, we define semicomputable and cosemicomputable algebras.

Definition 61. For Σ a finite S-sorted signature, a Σ-algebra A is *semicomputable* (respectively *cosemicomputable*) if there are an S-sorted recursive set W and an S-sorted map $f: W \to A$ such that the induced homomorphism $f^\#: G_\Sigma(W) \to A$ is surjective and the congruence $Q_{f^\#}$ is a recursively enumerable set (respectively its complement $G_\Sigma{}^2 - Q_{f^\#}$ is a recursively enumerable set). □

Thus, A semicomputable means that the word problem for A is *semidecidable*, i.e., there is an algorithm that assigns the value 1 to (t, t') iff $f^\#(t) = f^\#(t')$, but may not stop if $f^\#(t) \neq f^\#(t')$. Similarly, A cosemicomputable means that the word problem for A is *cosemidecidable*, i.e., there is an algorithm that assigns the value 0 to (t, t') iff $f^\#(t) \neq f^\#(t')$, but may not stop if $f^\#(t) = f^\#(t')$. Thus, an algebra is computable iff it is both semicomputable and cosemicomputable. Note that since $G_\Sigma \subseteq G_\Sigma(W)$ for any W, in the case where A is a minimal Σ-algebra, the above can be rephrased by saying that A is semicomputable (respectively cosemicomputable) iff the congruence Q_h associated to the unique homomorphism $h: G_\Sigma \to A$ is recursively enumerable (respectively its complement is recursively enumerable). Note also that the choice of the Gödel numbering $G_\Sigma(U)$ is immaterial, since it can be replaced by any other recursively isomorphic to it.

We have already pointed out that there are finitary initial algebra specifications with undecidable word problems [40] (however, they are semidecidable.)We now show that the final realization $N(M)$ of an M with computable behavior is always cosemicomputable, and we give later an example of an $N(M)$ that is undecidable.

Theorem 62. For Σ a finite signature, and V a subset of sorts, if an algebra M has a computable V-behavior, then its final realization $N(M)$ is cosemicomputable.

Proof. Since M has a computable V-behavior, there is a homomorphism $h: G_\Sigma(W) \to M$ surjective on the sorts in V, with $(Q_h)_v$ recursive for each v in V. Reasoning as in the proof of Theorem 59 there are recursive retracts $p_v: G_\Sigma(W)_v \to U_v$ and bijections $g_v: U_v \to M_v$ with $g_v \circ p_v = h_v$. Since being cosemicomputable is stable under isomorphism, we may assume without loss of generality that the V-sorted bijection $g: U \to M_V$ is the identity. We can then replace the above h by the homomorphism $\varepsilon_M: G_\Sigma(U) \to M$ for the purposes of the computability of the behavior. Indeed, consider the diagram of Figure 14.12, which shows $\varepsilon_M = h \circ j^\#$. Since the inclusion

$j: U \to G_\Sigma$ is a recursive function, $j^\#$ is a recursive homomorphism by Theorem 30, and thus $(Q_{\varepsilon_M})_v$ is recursive for each v in V. Note that, by the identification $U = M_V$ we have $(\varepsilon_M)_v = p_v \circ j^\#{}_v$. Thus, $(\varepsilon_M)_v$ is recursive for each v in V. Let t and t' be two terms in $G_\Sigma(U)$. We will be done if we exhibit a semidecision procedure for $\varepsilon_{N(M)}(t) \neq \varepsilon_{N(M)}(t')$ or equivalently for failure of $\varepsilon_{I(M)}(t)\text{ner}\varepsilon_{I(M)}(t')$. By definition of ner, this means that there is a v in V and a term u in $T_\Sigma(U \cup \{y\})_v$ such that (identifying $T_\Sigma(U)$ with $G_\Sigma(U)$),

$$\varepsilon_{I(M)}(u(y \leftarrow \varepsilon_{I(M)}(t))) \neq \varepsilon_{I(M)}(u(y \leftarrow \varepsilon_{I(M)}(t'))).$$

As in the proof of Theorem 19, this inequality can be rewritten as

$$\varepsilon_{I(M)}(u(y \leftarrow t)) \neq \varepsilon_{I(M)}(u(y \leftarrow t')).$$

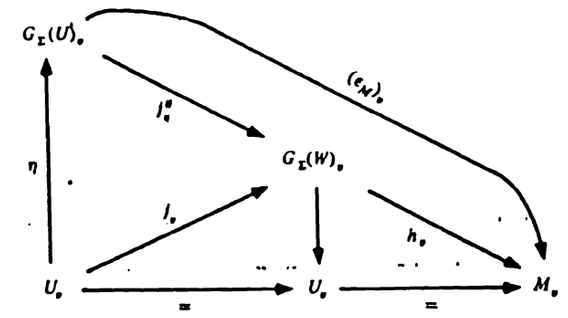
Since $I(M)$ is behaviorally identical to M this, in turn, can be written as

$$(*)\quad \varepsilon_M(u(y \leftarrow \varepsilon_{I(M)}(t))) \neq \varepsilon_M(u(y \leftarrow \varepsilon_{I(M)}(t'))),$$

Here then is the semidecision procedure: (i) number all the terms of $T_\Sigma(U \cup \{y\})_v$ for each v in V in a 'diagonalized' way, i.e., $u1$ of sort $v1, \ldots, un$ of sort vn, $un + 1$ of sort $v1$, etc.; and then (ii) compute (*) for each term $u = un$ (note that $(\varepsilon_M)_v$ is recursive); if there is a u giving an inequality, it will be found in a finite number of steps. □

At the end of Section 5 we pointed out that the most usual final realizations are those behaviorally equivalent to an initial algebra $T_{\Sigma,E}$ for E a set of equations, i.e., the behavior (for V a subset of sorts) is specified using initial algebra semantics, and then the final realization $N_{\Sigma,E}$ of that behavior is considered. This algebra is called the *final algebra* specified by (Σ, E) relative to the visible sorts V, or the final (Σ, E)-algebra relative to V. We also mentioned that $T_{\Sigma,E}$ need not coincide with the initial realization $I(T_{\Sigma,E})$, which does not have to satisfy E, but that there is a surjective strong V-homomorphism from $I(T_{\Sigma,E})$ to $T_{\Sigma,E}$. Thus even if the

Fig. 14.12



initial (Σ, E)-algebra is not computable, there is a computable realization, $I(T_{\Sigma,E})$, of its behavior, provided the behavior itself is computable.

Note also that we have proved that any V-reachable algebra M with a computable behavior is the Σ-reduct of a minimal algebra M' with a computable behavior for a finite enrichment without new sorts Σ' of its signature. Thus, it is natural to ask for a computational characterization of the class of machines having finitary final algebra specifications. We state below a conjecture on such a characterization, in a sense a converse to the last theorem. Call the V-behavior of an algebra M *nonunit* if there is a v in V such that M_v has more than one element.

Conjecture. For Σ a finite signature and V a subset of visible sorts, the following are equivalent for a minimal algebra M with nonunit computable V-behavior:

(i) M is cosemicomputable.

(ii) There is a finite enrichment without new sorts Σ' of Σ, and a finite set E of equations such that M is isomorphic to the reduct $N_{\Sigma',E|\Sigma}$ of the final (Σ', E)-algebra. □

We now give an example (inspired by [9]) of a final algebra with a computable behavior that is cosemicomputable but is not computable. The signature Σ is given in Figure 14.13.

There are no equations of sort **fun**; the equations of sort **nat** are the usual primitive recursive definitions of addition, $+$, truncated difference, $\dot{-}$ (i.e., $n \dot{-} m =$ **if** $n > m$ **then** $n - m$ **else** 0), multiplication, $\bullet$, the test for $n \neq 0$, and *min* (i.e., $min(n) =$ if $n = 0$ then 0 else 1), plus the following equations:

$$0[n1, \ldots, n14] = 0$$
$$1[n1, \ldots, n14] = 1$$
$$xi[n1, \ldots, n14] = ni \quad (\text{for } 1 \leq i \leq 14)$$
$$s(f)[n1, \ldots, n14] = s(f[n1, \ldots, n14])$$
$$(f1 + f2)[n1, \ldots, n14] = f1[n1, \ldots, n14] + f2[n1, \ldots, n14]$$
$$(f1 \dot{-} f2)[n1, \ldots, n14] = f1[n1, \ldots, n14] \dot{-} f2[n1, \ldots, n14]$$
$$(f1 \bullet f2)[n1, \ldots, n14] = f1[n1, \ldots, n14] \bullet f2[n1, \ldots, n14]$$
$$min(f)[n1, \ldots, n14] = min(f[n1, \ldots, n14]).$$

For E the above equations, $(T_{\Sigma,E})_{\mathbf{fun}} = (T_\Sigma)_{\mathbf{fun}}$, and $(T_{\Sigma,E})_{\mathbf{nat}}$ can be identified with ω. These equations evaluate each expression in the variables $x1, \ldots, xn$, to its result in ω after binding each xi to the value ni. Thus, it is clear that $T_{\Sigma,E}$ is computable; in particular, it has a computable behavior for $V = \{\mathbf{nat}\}$. (Incidentally, $T_{\Sigma,E}$ is the initial realization of its own behavior, since no nontrivial equations of sort **fun** can be deduced from E.)

Another very natural (Σ, E)-algebra is Ω with $\Omega_{\mathbf{nat}} = \omega$ and $\Omega_{\mathbf{fun}} = [\omega^{14} \to \omega]$, set of all functions of 14 variables on the natural numbers. The operation $\_[\_, \ldots, \_]$ is function evaluation, i.e., $f[n1, \ldots, n14] = f(n1, \ldots, n14)$; the operations s, $+$, $\dot{-}$, $\bullet$, and *min* are interpreted as usual on ω, and for the sort **fun** are interpreted as acting on the value of each function, i.e., are defined by the above equations; 0 and 1 are the constant functions with values 0 and 1 respectively, and xi is the ith projection function, i.e., is defined by the equation $xi[n1, \ldots, n14] = ni$ above. There is then a unique homomorphism h: $T_{\Sigma,E} \to \Omega$, and letting Ω_Σ denote the image subalgebra under this homomorphism, we obtain a minimal algebra behaviorally identical to $T_{\Sigma,E}$. We claim that Ω_Σ is the final (Σ, E)-algebra. To see this, note that if Ω_Σ were not final, there would be two functions $f1 \neq f2$, with corresponding expressions $t1$, $t2$, such that for each u in $T_\Sigma(\omega \cup \{y\})_{\mathbf{nat}}$ one would have

$$\varepsilon_{T_{\Sigma,E}}(u(y \leftarrow t1)) = \varepsilon_{T_{\Sigma,E}}(u(y \leftarrow t2)).$$

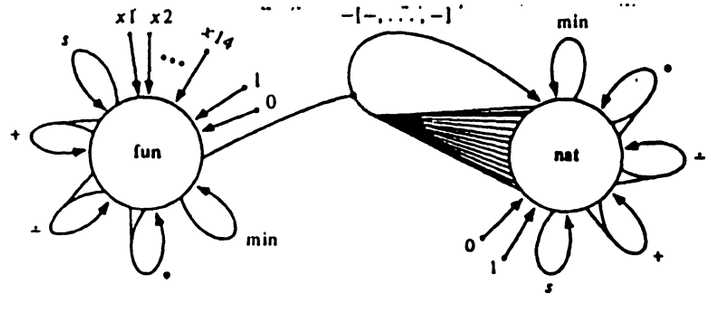
In particular, one would have $t1[n1, \ldots, n14] = t2[n1, \ldots, n14]$, for each $n1, \ldots, n14 \in \omega^{14}$, in contradiction to $f1 \neq f2$.

By Matijasevič's theorem [69, 70], Ω_Σ is not computable. Define an *ω-polynomial expression* (in 14 variables) to be a {**nat**}-irreducible term t of sort **fun** in $T_{\Sigma,E}$ such that the operations $\dot{-}$ and min do not occur in t (i.e., an expression on the variables $x1, \ldots, x14$ involving $+$, $\bullet$, and natural numbers as coefficients). Matijasevic's theorem can be formulated as follows:

Theorem 63. A set $U \subseteq \omega$ is recursively enumerable iff there are ω-polynomial expressions $t, t' \in T_{\Sigma,E}$ such that for each n in ω

$$n \in U \text{ iff } \exists m2, \ldots, m14 \in \omega \text{ such that } t[n, m2, \ldots, m14] = t'[n, m2, \ldots, m14] \text{ in } T_{\Sigma,E}.$$ □

Fig. 14.13. The signature of an example



It is a basic fact of recursive function theory that there are nonrecursive recursively enumerable sets, and that the complements of such sets are also not recursively enumerable. Let U be such a set. By Matijasevic's theorem there are ω-polynomials, t, t' such that for each number n, $n \in \omega - U$ iff

(*) $\forall m2, \ldots, m14 \in \omega$
$t[n, m2, \ldots, m14] \neq t'[n, m2, \ldots, m14]$ in $T_{\Sigma,E}$.

Let t_n be the ω-polynomial obtained by replacing each occurrence of $x1$ in t by $s^n(0)$. It is easy to see that for each $m1, \ldots, m14$ in ω one has

$$t_n[m1, \ldots, m14] = t[n, m2, \ldots, m14] \text{ in } T_{\Sigma,E}.$$

Define t'_n similarly from t'. Then condition (*) can be rephrased as

(**) $h(min(t_n \dot{-} t'_n) + (t'_n \dot{-} t_n))) = 1$ *in* Ω_Σ,

where h: $T_{\Sigma,E} \to \Omega_\Sigma$ is the unique homomorphism. If Ω_Σ were computable, we could decide for each n the word problem (**), i.e., we could decide $n \in \omega - U$, which is impossible.

This example shows the strong computational difference between intensional and extensional notions of function. Functions in *intensional* form (i.e., understood as *rules* of computation) are amenable to finitary specification by initial algebra semantics, whereas functions considered *extensionally* (i.e. identified as equal if they give the same result for all values) lead to cosemicomputable data types with a final algebra specification. The above example illustrated this for arithmetic expressions, but we could have chosen an entire programming language instead.

We now discuss a different notion of final algebra, due to Bergstra and Tucker, who have established a number of important theorems for this notion. We shall call their notion BT-final to avoid confusion, since the two notions are not equivalent; their intuition is also different, since there is no notion of visible sorts or of the behavior associated with a BT-final algebra (no sorts play a privileged role). Rather, their intuition is one of logical consistency. Before giving the definition, we will say a few words about *conditional equations*, i.e., equations of the form

$$(\forall X) \quad t1 = t'1 \ \& \ \ldots \ \& \ tn = t'n \Rightarrow t = t'.$$

An algebra A *satisfies* such an equation iff for any assignment f: $X \to A$ such that the conditions hold, the consequence also holds. Then A is a (Σ, E)-algebra, for E a set of conditional Σ-equations, iff A satisfies each equation in E; and a (Σ, E)-algebra is *initial* iff there is a unique Σ-homomorphism from it to any other (Σ, E)-algebra. Sound and complete many-sorted rules of deduction for conditional equations are given in [33]. These rules of deduction give the set E^* of all (ordinary) equations

that are satisfied by all (Σ, E)-algebras, for E a given set of conditional equations. [33] also shows that A is an initial (Σ, E)-algebra iff A is an initial (Σ, E^*)-algebra. Thus, T_{Σ,E^*} is also initial for the class of all algebras satisfying E. Similarly, by the *final* (Σ, E)-algebra $N_{\Sigma,E}$ (for V a subset of sorts), we mean the final realization $N(T_{\Sigma,E})$ of the V-behavior of the initial algebra $T_{\Sigma,E}$. We are now ready to define BT-final algebras.

Definition 64. Given a signature Σ and a set E of conditional equations, a BT-*final*-(Σ, E) algebra, if it exists, is a minimal (Σ, E)-algebra F such that, if h: $T_\Sigma \to F$ is the unique homomorphism, then the following hold:

(i) $Q_h \neq T_\Sigma^2$ (i.e., F is not the 'unit' algebra).
(ii) If $h(t) \neq h(t')$, then $T_{\Sigma, E \cup \{\forall \emptyset \, t = t'\}}$ is the unit algebra, i.e. it has exactly one point of sort s if $(T_\Sigma)_s$ is nonempty. □

Thus, the BT-final algebra is the algebra obtained by imposing on $T_{\Sigma,E}$ *all* the equations $t = t'$ such that there is a nonunit minimal $(\Sigma, E \cup \{t = t'\})$-algebra, i.e., all equations that in a certain sense are not 'inconsistent' with the equations E. In general such a process, although well-defined, may yield an algebra that is the unit algebra. For example, with the natural numbers, the equations $2 = 0$ and $3 = 0$ have nonunit models, but the two together collapse all the natural numbers to one point. But, when they do exist, any two BT-final algebras are isomorphic and can be characterized as the final object of the category with objects minimal (Σ, E)-algebras (with exclusion of the unit algebras) and morphisms the Σ-homomorphisms. Thus, all the final BT-algebras, if they exist, form an abstract data type, and we talk of *the* BT-final (Σ, E)-algebra.

Here is the theorem of [9] characterizing cosemicomputable algebras; their proof uses Matijasevic's theorem.

Theorem 65. Let A be a minimal Σ-algebra (Σ finite and one-sorted). Then the following are equivalent:

(i) A is cosemicomputable.
(ii) A is the Σ-reduct of a BT-final (Σ', E)-algebra, for Σ' an enrichment without new sorts of Σ by at most 5 hidden functions, and E a set of at most $15 + |\Sigma|$ conditional equations.

The same holds for A a minimal Σ-algebra when Σ is finite and many-sorted, making the appropriate modifications on the bounds for the number of hidden functions and conditional equations. □

We conclude this section with a very nice theorem of Bergstra & Tucker

[8] showing the simultaneous adequacy of initial and BT-final algebra semantics to specify computable algebras, and giving a bound on the number of hidden functions and equations required. This bound depends only on the number of sorts and not on the size of the signature Σ.

Theorem 66. Let A be a minimal Σ-algebra, and n the number of sorts of its signature Σ. Then the following are equivalent:

(i) A is computable.

(ii) A is the Σ-reduct of an algebra that is both initial and BT-final for (Σ', E) an enrichment without new sorts having at most $3(n+1)$ hidden functions and $2(n+1)$ new equations. □

The proof of this theorem also makes essential use of Matijasevic's theorem. The original algebra is 'rigidified' by introducing new operations that act as injections (with corresponding retractions) of each sort into a highest cardinality sort. Identification of any two elements after this enrichment produces the unit algebra. Further enrichment, use of Matijasevic's theorem, and an elegant 'folding' of equations using conditionals give the required result and bounds.

6.7 *Equality enrichments, computability and inductionless induction*

Whatever other operations an abstract data type may have, programming intuition strongly suggests that it can be given *equality* operations that tell whether or not two abstract data items are the same; intuition also suggests that these operations will be equationally definable [31]. This subsection gives a formal justification to this intuition by showing that a data type is computable if and only if its equality can be axiomatized with a finite number of equations. This can be seen as a purely algebraic formulation of a Church-like thesis, that the intuitive notion of computability agrees with certain algebraic concepts. The equational axiomatization of equality is also closely related to the recent theorem-proving method called 'inductionless induction', which uses purely equational reasoning (in the form of rewrite-rules) to prove theorems valid in an initial algebra that would normally have to be proved by induction. We explain the basic facts about the satisfaction of equations in initial algebras and about inductionless induction, and give pointers to further developments in this area. This subsection drops the implicit assumption of finiteness for signatures.

Intuitively, we seek to enrich a given data type with equality predicates, i.e., operations $\equiv_s : ss \to$ **newbool** for each sort s, where **newbool** is a new sort with constants *true* and *false*, we also want to *axiomatize* those operations by giving new equations such that, for any two ground terms t, t', one can prove $(t \equiv t') = true$ (respectively *false*) iff t and t' can (respectively cannot) be proved equal in our original data type by the rules of equational deduction, and, of course, one cannot prove *true* = *false*. For instance the equations

$$(x \equiv x) = true$$
$$(0 \equiv s(x)) = false$$
$$(s(x) \equiv 0) = false$$
$$(s(x) \equiv s(y)) = (x \equiv y),$$

give such an axiomatization for the natural numbers.

We also desire that the new equations should have no effect on the old sorts. This property is meaningful for any enrichment, and corresponds to sufficient completeness plus consistency in Guttag's terminology; it is weaker than 'persistence' since it is only stated for the initial algebra.

Definition 67. Given an enrichment (Σ', E') of (Σ, E), there is a unique Σ-homomorphism $h: T_{\Sigma,E} \to T_{\Sigma',E'}|_\Sigma$. Then this enrichment is *protected* iff h is an isomorphism. □

We give now the definition of an 'equational equality presentation'; the definition is meaningful even without explicitly giving a subpresentation that it enriches by equality. In case this is explicit, the equational equality presentation is called an 'equality enrichment' of the given subpresentation.

Definition 68. Let $\Sigma^=$ be a signature that contains a sort **newbool** with constants *true* and *false*, and for each sort $s \neq$ **newbool** an operation $\equiv_s : ss \to$ **newbool**; let $E^=$ be a set of $\Sigma^=$-equations. Then $(\Sigma^=, E^=)$ is an *equational equality presentation* iff it satisfies the following conditions:

(1) *Equational equality.* For each sort $s \neq$ **newbool** in $\Sigma^=$ and each t, t' in $(T_{\Sigma^=})_s$:

a. The equation $(\forall \varnothing)\ (t \equiv t') = true$ is provable from $E^=$ if and only if $(\forall \varnothing)\ t = t'$ is provable from $E^=$.

b. The equation $(\forall \varnothing)\ (t \equiv t') = false$ is provable from $E^=$ if and only if $(\forall \varnothing)\ t = t'$ is *not* provable from $E^=$.

(2) *Consistency.* It is not provable from $E^=$ that $(\forall \varnothing)\ true = false$ for sort **newbool**.

In addition, if there is a subpresentation (Σ, E) with sorts those of Σ except **newbool** and such that the enrichment $(\Sigma, E) \subseteq (\Sigma^{=}, E^{=})$ is protected, then we will call $(\Sigma^{=}, E^{=})$ an *equality enrichment* of (Σ, E). Note that by having a protected enrichment, the equational equality condition is then equivalent to the following:

(1') *Equational equality'*. For each sort s of Σ and each t, t' in $T_{\Sigma,s}$:
   a. The equation $(\forall\varnothing)\ (t \equiv t') = \mathit{true}$ is provable from $E^{=}$ if and only if $(\forall\varnothing)\ t = t'$ is provable from E.
   b. The equation $(\forall\varnothing)\ (t \equiv t') = \mathit{false}$ is provable from $E^{=}$ if and only if $(\forall\varnothing)\ t = t'$ *is not* provable from E.

In other words, the equality predicate in the enrichment is characterized by equational deduction in the original subspecification. □

In general, the property of an enrichment being protected requires careful analysis. However, for the case of equality enrichments there is a simple sufficient condition that applies to all reasonable situations that appear in practice:

Lemma 69. Let $(\Sigma, E) \subseteq (\Sigma', E')$ be an enrichment such that: there is only one sort s_0 in Σ' and not in Σ; the operations and constants in Σ' that are not in Σ all have sort s_0; and the equations in E' that are not in E all have sort s_0. Then the enrichment is protected.

Proof. For each sort $s \neq s_0$ we have $T_{\Sigma,s} = T_{\Sigma',s}$ and, by inspecting the rules of many-sorted equational deduction, it is easy to check that since there are no operations of sort different from s_0 which have s_0 as an argument, the equations in E' and not in E have no effect whatsoever on terms of sort different from s_0. □

A close connection exists among initial, final and BT-final algebras for equational equality presentations.

Lemma 70. Let $(\Sigma^{=}, E^{=})$ be an equational equality presentation. Then:

(1) Taking $V = \{\mathbf{newbool}\}$ as the set of visible sorts, the initial and the final $(\Sigma^{=}, E^{=})$-algebras coincide.
(2) Assuming that $(T_{\Sigma^{=},E^{=}})_{\mathbf{newbool}} = \{[\mathit{true}], [\mathit{false}]\}$ and adding to $E^{=}$ the conditional equation
$\forall\{x, y\}\ \ \mathit{true} = \mathit{false} \Rightarrow x = y$
for each sort different from newbool to form an enrichment $(\Sigma^{=}, E^{=\prime})$, the initial $(\Sigma^{=}, E^{=\prime})$-algebra and the BT-final $(\Sigma^{=}, E^{=\prime})$-algebra coincide.

Proof. To prove (1), assume $[t] = [t']$ in the final algebra but $[t] \neq [t']$ in the initial algebra. This gives $[\mathit{true}] = [t \equiv t] = [t \equiv t'] = [\mathit{false}]$ in the final algebra, a contradiction.

To prove (2), first note that $(T_{\Sigma^{=},E^{=}})$ vacuously satisfies the conditional equation of (2), i.e., it is also initial $(\Sigma^{=}, E^{=\prime})$-algebra. We will be done if we show that the only proper quotient, A, of $T_{\Sigma^{=},E^{=}}$ that satisfies $E^{=}$ is the unit algebra. Indeed, for such an A, if $[t] = [t']$ in A and if $[t] \neq [t']$ in the initial algebra, then

(i) If the sort is newbool this means that $[\mathit{true}] = [\mathit{false}]$, and hence everything reduces to one point for the sort **newbool**, and so by the conditional equations, everything also reduces to one point in any other nonempty sort; i.e., A is the unit algebra.
(ii) For any other sort, we reason as in the proof of (1) and reduce to the case (i). □

The following theorem characterizes the computability of an abstract data type in terms of equational equality and initiality. From the last lemma, one can obtain as immediate corollaries two similar characterizations replacing initiality by either finality or by BT-finality.

Theorem 71. For Σ a finite signature and a minimal Σ-algebra the following are equivalent:

(1) A is computable.
(2) There is a finite enrichment $\Sigma \subseteq \Sigma^{=}$ with only one new sort newbool and a finite set $E^{=}$ of $\Sigma^{=}$-equations such that $(\Sigma^{=}, E^{=})$ is an equational equality presentation and A is Σ-isomorphic to the reduct $(T_{\Sigma^{=},E^{=}})|_{\Sigma}$.
(3) Same as (2) plus the equations $E^{=}$ are confluent and terminating as rewrite-rules.

Proof. Clearly, (3) ⇒ (2). To see (2) ⇒ (1), notice that we can decide the word problem for A by the following algorithm: given ground Σ-terms t and t', start generating all the consequences of $E^{=}$ by the rules of many-sorted equational deduction. After a finite number of steps you either obtain the equation $(\forall\varnothing)\ t = t'$ (if $t = t'$ in A), or the equation $(\forall\varnothing)\ (t \equiv t') = \mathrm{false}$ (if $t \neq t'$ in A).

To see (1) ⇒ (3), we may assume without loss of generality that A is a recursive algebra. We take $\Sigma^{=0}$ the enrichment of Σ by sort newbool with constants true and false and operations $\equiv_s: ss \to s$ for each old sort s. We extend A to a recursive $\Sigma^{=0}$-algebra $A^=$ in the obvious way: sorts and operations for the signature Σ are those of A, $(A^=)_{\text{newbool}} = \{0, 1\}$ with false = 0 and true = 1; for each sort s in Σ, $(A^=)_{=_s}$ is the function:

$\lambda(x, y)$. if $x = y$ then 1 else 0,

which is clearly recursive. By Theorem 54 characterizing computable algebras by rewrite rules, we know that there is a finite enrichment without new sorts $\Sigma^{=0} \subseteq \Sigma^=$ and a finite set $E^=$ of usable equations such that the induced rewriting relation $\to$ is terminating and confluent, and $A^=$ is isomorphic to the $\Sigma^{=0}$-reduct of the initial algebra $T_{\Sigma^=, E^=}$. As a consequence, A is isomorphic to the Σ-reduct of that initial algebra. To finish the proof we need only note that $(\Sigma^=, E^=)$ is an equational equality specification. The consistency property is clear, and the equational equality property follows from the bijection between $A^=$ and $T_{\Sigma^=, E^=}$ and the definition of the equality predicates $(A^=)_{=_s}$. □

We will now consider the relationship between equality enrichments and the satisfaction of equations in initial algebras. This relationship, namely the reduction of satisfaction to consistency, underlies the 'inductionless induction' theorem proving method. Finally, we briefly discuss the literature in this area.

An initial (Σ, E)-algebra in general satisfies more equations than just those deducible from E by the rules of equational deduction. For example, the natural numbers with zero, successor and addition are the initial algebra for the following equations E:

$$x + 0 = x$$
$$0 + x = x$$
$$s(x) + y = s(x + y)$$
$$x + s(y) = s(x + y),$$

and it is well-known that natural number addition satisfies the associative law

$$(x + y) + z = x + (y + z).$$

However, this law is not satisfied by all the algebras that satisfy the above equations E. One way to see this is to first remark that the above rules are indeed terminating and confluent (more justification for this below), and they remain terminating and confluent when T_Σ is replaced by $T_\Sigma(X)$ for X a set of additional constants, and then give rise to a canonical term algebra $\text{Can}_{\Sigma,E}(X)$ which is an initial $(\Sigma(X), E)$-algebra; but for $X = \{a, b, c\}$, $\text{Can}_{\Sigma,E}(X)$ does *not* satisfy the associativity law, since the terms $(a + b) + c$ and $a + (b + c)$ are both in canonical form.

Associativity of + depends on the additional fact that the natural numbers are initial. This must be used in any proof of associativity by induction. We will now establish two basic lemmas about the satisfaction of equations in initial algebras.

Lemma 72. Let $(\Sigma, E) \subseteq (\Sigma, E')$ be an enrichment by equations only. Then the initial algebra $T_{\Sigma,E}$ satisfies the equations in E' iff the enrichment is protected, i.e., iff $T_{\Sigma,E} = T_{\Sigma,E'}$.

Proof. If $T_{\Sigma,E} = T_{\Sigma,E'}$ then $T_{\Sigma,E}$ clearly satisfies E'. Conversely, if $T_{\Sigma,E}$ satisfies E' then there is a unique homomorphism $j: T_{\Sigma,E'} \to T_{\Sigma,E}$. Now since $T_{\Sigma,E'}$ certainly satisfies $E \subseteq E'$, there is a unique homomorphism $q: T_{\Sigma,E} \to T_{\Sigma,E'}$. Then j and q must be isomorphisms, since they give rise to endomorphisms $j \circ q$ and $q \circ j$ that by initiality must satisfy $j \circ q = 1_{T_{\Sigma,E}}$ and $q \circ j = 1_{T_{\Sigma,E'}}$. Indeed, q and j are both identity functions, since again by initiality, $q \circ h = h'$, for h, h' the unique homomorphisms from T_Σ so that the congruences associated to h and h' are identical, i.e., $T_{\Sigma,E} = T_{\Sigma,E'}$. □

Lemma 73. A set E'' of Σ-equations holds for the initial (Σ, E)-algebra iff it holds for the initial (Σ', E')-algebra for *every* protected enrichment $(\Sigma, E) \subseteq (\Sigma', E')$.

Proof. If E'' holds for every protected enrichment, it will in particular hold for the trivial one.

Conversely, let $(\Sigma, E) \subseteq (\Sigma', E')$ be an arbitrary protected enrichment and let $(\forall X)\ t = t'$ be an equation in E not satisfied by the initial (Σ', E')-algebra. This means that if X consists of variables $x_1, \ldots, x_n$, there is an assignment $f: X \to T_{\Sigma',E'}$, say $f(x_i) = [t_i]$, such that $f^\#(t) \neq f^\#(t')$. Since the enrichment $(\Sigma, E) \subseteq (\Sigma', E')$ is protected, we may assume that the representatives $t_1, \ldots, t_n$ are Σ-terms. This provides a similar assignment $f^0: X \to T_{\Sigma,E}$ by $f^0(x_i) = [t_i]$ from which (using protection of the enrichment and the obvious factorization of $f^\#|_{T_\Sigma(X)}$ as $(f^0)^\#$ composed with the isomorphism $h: T_{\Sigma,E} \to T_{\Sigma',E'}|_\Sigma$ which follows from initiality) one sees that the equation $(\forall X)\ t = t'$ does not hold in $T_{\Sigma,E}$. □

We are now ready to reduce the problem of satisfaction of a set of equations in an initial algebra to that of consistency in an equality enrichment augmented by those equations:

Theorem 74. Let (Σ, E) be a presentation and let $(\Sigma^{=}, E^{=})$ be an equality enrichment of it. Then a set E' of Σ-equations is satisfied by the initial (Σ, E)-algebra if and only if $(\forall \varnothing)\ true = false$ is not deducible from $E^{=} \cup E'$ by the rules of many-sorted equational deduction.

Proof. If E' holds for the initial (Σ, E)-algebra then, by the previous lemma, it also holds for the protected enrichment $(\Sigma^{=}, E^{=})$; it then follows from Lemma 72 and the consistency property of the equality enrichment that $(\forall \varnothing)\ true = false$ is not deducible from $E^{=} \cup E'$.

Conversely, suppose that an equation $(\forall X)\ t = t'$ in E', say of sort s, is not satisfied by $T_{\Sigma,E}$; i.e., suppose that there is an assignment $f: X \to T_\Sigma$ such that $[f^\#(t)] \neq [f^\#(t')]$ in $T_{\Sigma,E}$; then

(i) $(\forall \varnothing)\ (f^\#(t) \equiv_s f^\#(t')) = false$

can be deduced from $E^{=}$ by the rules of many-sorted equational deduction. On the other hand

(ii) $(\forall \varnothing) f^\#(t) = f^\#(t')$

can be deduced from E' by the rules of equational deduction, and by reflexivity we have

(iii) $(\forall \{x, y\})\ (x \equiv_s y) = (x \equiv_s y)$.

Hence from (ii) and substitutivity we can deduce

(iv) $(\forall \{x\})\ (x \equiv_s f^\#(t)) = (x \equiv_s f^\#(t'))$.

Again by substitutivity and reflexivity we can then deduce that

(v) $(\forall \varnothing)\ (f^\#(t) \equiv_s f^\#(t)) = (f^\#(t) \equiv_s f^\#(t'))$.

Since $f^\#(t) \equiv_s f^\#(t)) = true$ follows from $E^{=}$ by the rules of deduction, from $E^{=} \cup E'$ and transitivity we deduce $(\forall \varnothing)\ (f^\#(t) \equiv_s f^\#(t')) = true$, which together with (i) gives $(\forall \varnothing)\ true = false$, □

This theorem provides an 'inductionless' (i.e., purely equational) way of proving that an initial algebra satisfies a given equation. Several algorithms can help in automating most of the proof effort, turning it into a theorem-proving strategy. On the one hand, the Knuth–Bendix algorithm can attempt to find a set of confluent equations deductively equivalent to a given set of equations, provided termination is satisfied; on the other, attempts to prove rewrite-rule termination can also be semiautomated [32] or even automated [52]. Here then is a possible strategy, using the Knuth–Bendix algorithm, to prove that a set E' of equations holds for the initial (Σ, E)-algebra:

(i) Enrich (Σ, E) to a confluent and terminating equality enrichment $(\Sigma^{=}, E^{=})$.

(ii) Use Knuth–Bendix and termination methods to attempt completing $E^{=} \cup E'$ to a confluent and terminating set of equations.

(iii) If somewhere in the completion process for $E^{=} \cup E'$ the equation $(\forall \varnothing)\ true = false$ is derived, then stop: at least one of the equations in E' is not satisfied by $T_{\Sigma,E}$.

(iv) If the completion process terminates with a set of confluent and terminating rewrite rule for which $[true] \neq [false]$, then the equations E' are satisfied by $T_{\Sigma,E}$.

(v) Otherwise (i.e., if the completion process does not terminate and we could not prove $(\forall \varnothing)\ true = false$ from already generated rules), nothing can be *decided* about the satisfaction of the equation. Nevertheless, if we were to ideally 'wait forever', this would actually give a proof that the equation holds; this is so because, in the limit, the set of all generated equations is confluent [43].

For example, consider the associativity of natural number addition. The set $E^{=}$ below is a confluent and terminating equality enrichment:

$x + 0 = x$
$0 + x = x$
$s(x) + y = s(x + y)$
$x + s(y) = s(x + y)$
$(x \equiv x) = true$
$(0 \equiv s(x)) = false$
$(s(x) \equiv 0) = false$
$(s(x) \equiv s(y)) = (x \equiv y)$.

It so happens that $E^{=}$ union with the equation

$(x + y) + z = x + (y + z)$

is already terminating and confluent, and is certainly consistent (i.e., $true \neq false$), so that associativity follows. Termination can be seen using the following ordering on terms: $t \leq t'$ iff $\phi(t) \leq \phi(t')$ where $\phi(0) = \phi(true) = \phi(false) = 1$, $\phi(x) = 1$ for any variable x, and $\phi(s(t)) = \phi(t) + 1$, $\phi(t + t') = \phi(t) \bullet 3^{\phi(t')}$, and $\phi(t \equiv t') = \phi(t) + \phi(t') + 1$. Then one can see that for any one step rewriting $t \to t'$ induced by the equations, $\phi(t') < \phi(t)$; hence the rules are terminating. Confluence is handled semiautomatically by the Knuth–Bendix algorithm, which for the above equations stops without producing any new rules. This is because all critical pairs produce the same normal form. For instance, the associativity equation and the equation $x + s(y) = s(x + y)$ give the critical pair $\langle x + (y + s(z)); s((x + y) + z)\rangle$, both one step rewritings from $(x + y) + s(z)$, and both sides rewrite to $s(x + (y + z))$. We leave the reader to

actually compute all cases (see [42] for a precise definition of critical pair).

The inductionless induction method is originally due to Musser [72]. Goguen [31] generalized and simplified the method, and proved Theorem 74. Huet & Hullot [44] give a variant of the method that when certain conditions are satisfied by the original equations, does not require the introduction of an equality predicate; intuitively, if there is a subsignature $\Omega \subseteq \Sigma$ of 'constructors' (with same set of sorts) such that the enrichment $(\Omega, \varnothing) \subseteq (\Sigma, E)$ is protected, i.e., such that each equivalence class $[t]$ of Σ-terms has a unique Ω-term as its representative, then we can handle equality implicitly, as identity between the representative Ω-terms. This idea has been extended further to the case of a protected enrichment $(\Omega, E_0) \subseteq (\Sigma, E)$ by Kirchner [54]; this opens generalizations of the above method that use generalized Knuth–Bendix algorithms modulo 'nice' equations such as associativity and commutativity for proofs by inductionless induction; termination methods in this context have recently been considered [23]. Lankford [56] discusses potential limitations of the inductionless induction method, and [52] gives a careful explanation and examples of the method (for the case without equality predicates).

6.8 *Concluding remarks on abstract data type computability*

This brief subsection indicates some additional references and research directions in abstract data type computability; it claims neither exhaustion nor completeness.

6.8.1 *The classics*

Even before the establishment of any formal notions of computability, van der Waerden [85] defined 'explicitly given fields' and proved [86] that there was no general splitting algorithm applicable to all explicitly given fields. The subject of computable fields was further developed in the framework of computability theory by Frölich & Shepherdson [24] and later by Rabin [78], who proved that the algebraic closure of a computable field is also computable. Both Rabin [78] and Malcev [67] develop equivalent versions of computable algebra for an arbitrary signature, as in Section 6.3, and establish the foundations of the subject.

6.8.2 *Further work by Bergstra, Tucker et al.*

(i) By Bergstra and Tucker, besides the references already cited, see [7, 10, 11, 12].
(ii) Asveld & Tucker [1] study the computational complexity of abstract data types.
(iii) Bergstra, Broy, Tucker & Wirsing [14] give characterization theorems for hierarchical specifications and partial abstract data types.
(iv) Bergstra & Klop [3, 4] begin the subject of computability of parameterized abstract data types.

6.8.2 *Computability of partial abstract data types with equationally defined domains*

A natural way of extending (total) data types is to consider partial data types with operations defined on (vectors of) values that satisfy equational conditions (e.g., empty(x) = false). This approach has been proposed in [79]. Kaphengst [53] gives a careful study of the computability of these data types, and Hupbach [47] studies the related problem of computability for implementations.

Acknowledgements

We would very much like to thank Rod Burstall, David Plaisted, and Rob Shostak for their comments on an early draft of this paper. Special thanks to Arthur Knoebel, Melissa Smartt, and other members of the New Mexico State University Theoretical Computer Science Seminar, to T. Rus, and to Jean-Pierre Jouannaud for their detailed comments and suggested improvements on a more recent version. Andrew Black, Stephen Bloom, Akira Kanda, Fernando Orejas and Bill Wadge provided comments and pointed out mistakes or possible improvements; to each of them we also express our thanks. Responsibility for any mistakes is, of course, entirely ours.

Appendix: proofs of soundness and completeness

We first prove the Soundness Theorem stated in Section 4.3.3.

Proof of Theorem 12. For technical reasons, it is easier to prove the

soundness of a set of rules equivalent to those given in Section 4.3.2. The new rules are (1)–(3), as before, together with:

(4′) *Substitutivity–1.* If

$(\forall Y)\, u1 = u2$

of sort s is derivable and if $\sigma \in \Sigma_{s1\ldots sn,s'}$ is an operation with $sk = s$, then so is

$(\forall Z)\ \sigma(x1, \ldots, xk-1, u1, xk+1, \ldots, xn) = \sigma(x1, \ldots, xk-1, u2, xk+1, \ldots, xn)$,

where $Z = Y \cup \{x1, \ldots, xk-1, xk+1, \ldots, xn\}$ with xj of sort sj.

(5′) *Substitutivity–2.* If

$(\forall X)\, t = t'$

is derivable and if $g: X \to T_\Sigma(Y)$ is an S-sorted map, then

$(\forall Y)\, g^\#(t) = g^\#(t')$

is also derivable.

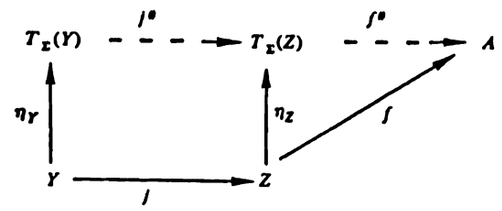
We first prove the soundness of the rules (1)–(3), (4′), (5′), and then prove their equivalence with the original rules (1)–(6) in a subsequent lemma. Soundness of rules (1)–(3) follows directly from the definition of satisfaction and is left to the reader; the soundness of rules (4′) and (5′) remains. For (4′), we must show that if A is an algebra satisfying $(\forall Y)\ u1 = u2$, then A also satisfies

$(\forall Z)\ \sigma(x), \ldots, xk-1, u1, xk+1, \ldots, xn) = \sigma(x1, \ldots, xk-1, u2, xk+1, \ldots, xn)$.

Now let $f: Z \to A$ be a (S-sorted) map, and consider the commutative diagram in Figure 14.14, where $j: Y \to Z$ is the inclusion map which induces the inclusion homomorphism $j^\#: T_\Sigma(Y) \to T_\Sigma(Z)$. We then have

$(f^\#(\sigma(x1, \ldots, xk-1, u1, xk+1, \ldots, xn))$
$= \sigma(f^\#(x1), \ldots, f^\#(xk-1), f^\#(u1), f^\#(xk+1), \ldots, f^\#(xn))$ (by $f^\#$ a homomorphism)
$= \sigma(f^\#(x1), \ldots, f^\#(xk-1), f^\#(j^\#((u1)), f^\#(xk+1), \ldots, f^\#(xn))$
$= \sigma(f^\#(x1), \ldots, f^\#(xk-1), (f \circ j)^\#(u1), f^\#(xk+1), \ldots, f^\#(xn))$ (by $j^\#$ inclusion and diagram above)

Fig. 14.14



$= \sigma(f^\#(x1), \ldots, f^\#(xk-1), (f \circ j)^\#(u2), f^\#(xk+1), \ldots, f^\#(xn))$ (by hypothesis)
$= f^\#(\sigma(x1, \ldots, xk-1, u2, xk+1, \ldots, xn))$ (reversing the steps)

as desired.

To see the soundness of (5′), let A satisfy $(\forall X)\ t = t'$, let $f: Y \to A$ and $g: X \to T_\Sigma(Y)$ be maps. Then the diagram of Figure 14.15 shows that $f^\# \circ g^\# = (f^\# \circ g)^\#$, and so we have that

$f^\#(g^\#(t)) = (f^\# \circ g)^\#(t) = (f^\# \circ g)^\#(t')$ (by hypothesis)
$= f^\#(g^\#(t'))$,

as desired. To finish the proof we need only prove

Lemma 75. The rules (1)–(3), (4′), (5′) are equivalent to the rules (1)–(6), i.e., an equation $(\forall X)\ t = t'$ is derivable by the first set of rules from a set E of equations iff it is derivable by the second set of rules from the same set of equations.

Proof of Lemma. For the 'if' part, we must show that any equation derivable by the rules (4)–(6) can be derived using (1)–(3), (4′), (5′). First note that (5) and (6) are particular instances of (5′): for (5), take as g the inclusion $X \to X \cup \{y\} \to T_\Sigma(X \cup \{y\})$; for (6), take $g: X \to T_\Sigma(X - \{x\})$ with $g(x') = x'$ if $x' \neq x$, and $g(x) = v \in (T_\Sigma)_s$. For (4), reason by induction on $n = \max(\text{depth}(t1), \text{depth}(t2))$ where $\text{depth}(t) = 0$ if t is a variable or a constant and $\text{depth}(\sigma(v1, \ldots, vm)) = 1 + \max\{\text{depth}(v1), \ldots, \text{depth}(vm)\}$. We leave the reader to check the case $n = 0$. Let $n + 1 = \max\{\text{depth}(t1), \text{depth}(t2)\}$; say $n + 1 = \text{depth}(t1)$, $t1 = \sigma(v1, \ldots, vm)$. Then we have

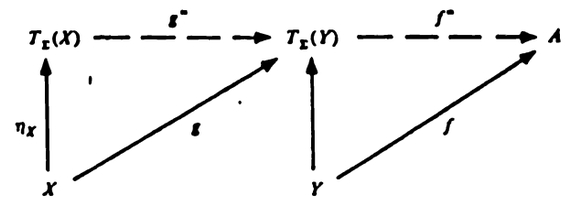
$t1(x \leftarrow u1) = \sigma(v1(x \leftarrow u1), \ldots, vm(x \leftarrow u1))$
$= \sigma(v1(x \leftarrow u2), \ldots, vm(x \leftarrow u2))$

since by induction hypothesis, $(\forall Z)\ vi(x \leftarrow u1) = vi(x \leftarrow u2)$ can be derived using (1)–(3), (4′), and (5′). Then by m applications of (4′) we have

$= t1(x \leftarrow u2)$
$= t2(x \leftarrow u2)$ (by (5′)),

as desired.

Fig. 14.15



For the 'only if' part, we must show that any equation derivable using rules (4') or (5') can be derived using rules (1)–(6). Rule (4') is a particular instance of (4) with $t1 = t2 = \sigma(x1, \ldots, xn)$ and with $x = xk$. Rule (5') follows: (i) if $X = \varnothing$, from as many applications of the rule of abstraction as variables in Y; (ii) if $X \neq \varnothing$, from as many applications of the rule of substitutivity as variables in X. □□

Now we prove the Completeness Theorem stated in Section 4.3.3.

Proof of Theorem 13. We have to show that if an equation

(*) $(\forall X)\, t = t'$

is satisfied by all (Σ, E)-algebras, then it is derivable from E using the rules of deduction (1)–(6) or by the above lemma, using the equivalent rules (1)–(3), (4'), and (5'). Assume that the equation (*) is satisfied by all (Σ, E)-algebras, but is not derivable. We will reach a contradiction by considering the algebra $T_{\Sigma,E}(X)$, defined as the quotient of $T_\Sigma(X)$ by the congruence $E^\#$ such that (u, v') is in $E^\#$ iff $(\forall X)\, u = v$ is derivable from E using the rules (1)–(3), (4'), and (5'). The fact that $E^\#$ is a congruence follows trivially from the rules (1)–(3) and (4'). Also $T_{\Sigma,E}(X)$ is a (Σ, E)-algebra, since for any equation $(\forall Y)\, u = u'$ in E with, say $Y = \{y1, \ldots, yn\}$, and for $f\colon Y \to T_{\Sigma,E}(X)$ a map with, say $f(yj) = tj$, we have

$$\begin{aligned} f^\#(u) &= u(y1 \leftarrow t1, \ldots, yn \leftarrow tn) \\ &= u'(y1 \leftarrow t1, \ldots, yn \leftarrow tn) \quad \text{(by } n \text{ applications of the rule (5'))} \\ &= f^\#(u'), \end{aligned}$$

as desired. By hypothesis the equation (*) holds for all (Σ, E)-algebras but is not derivable. This means that $[t] \neq [t']$ in $T_{\Sigma,E}(X)$, which contradicts the fact that (*) is satisfied in $T_{\Sigma,E}(X)$, so in particular $[t] = [t']$ when we consider the assignment

$$X \to T_\Sigma(X) \to T_{\Sigma,E}(X)$$

obtained by composing the inclusion of X with the quotient map from $T_\Sigma(X)$ to $T_{\Sigma,E}(X)$. □

References

1 Asveld, P. R. J., and Tucker, J. V. Complexity theory and the operational structure of algebraic programming systems. *Acta Informatica*, 17, 451–76, 1982.

2 Benabou, J. Structures Algébriques dans les Categories. *Cahiers de Topologie et Géometrie Differentiel*, 10, 1–126, 1968.

3 Bergstra, J. A., and Klop, J. W. *Algebraic specifications for parameterized data types with minimal parameter and target algebras.* Technical Report IW 183, Mathematical Center, Dept. of Computer Science, Amsterdam, 1981. 22 pp.

4 Bergstra, J. A., and Klop, J. W. *Initial algebra specifications for parameterized data types.* Technical Report IW 186, Mathematical Center, Dept. of Computer Science, Amsterdam, 1981. 20 pp.

5 Bergstra, J. A. and Meyer, J.-J. I/O-Computable Data Structures. *SIGPLAN Notices*, 16(4), 27–32, 1981.

6 Bergstra, J. A. and Tucker, J. V. *A characterization of computable data types by means of a finite, equational specification method.* Technical Report, Mathematisch Centrum, Amsterdam, Holland, 1979. Preprint IW 124/79, November, 1979.

7 Bergstra, J. A., and Tucker, J. V. On the adequacy of finite equatorial methods for data type specification. *SIGPLAN Notices*, 14(11), 13–18, 1979.

8 Bergstra, J. A., and Tucker, J. V. *The completeness of the algebraic specification methods for data types.* Technical Report IW 156, Mathematical Center, Dept. of Computer Science, Amsterdam, 1980. 18 pp.

9 Bergstra, J. A., and Tucker, J. V. *Initial and final algebra semantics for data type specifications: two characterisation theorems.* Technical Report IW 142, Mathematical Centre, Dept. of Computer Science, Amsterdam, 1980. To appear in *SIAM Journal on Computing*, 36 pp.

10 Bergstra, J. A., and Tucker, J. V. A natural data type with a finite equational final semantics specification but no effective equational initial semantics specification. *Bulletin European Association for Theoretical Computer Science*, 11, 23–33, 1980.

11 Bergstra, J. A., and Tucker, J. V. *Equational specifications for computable data types: 6 hidden functions suffice and other sufficiency bounds.* Technical Report IW 128, Mathematical Centre, Dept. of Computer Science, Amsterdam, 1980. 16 pp.

12 Bergstra, J. A., and Tucker, J. V. *On bounds for the specification of finite data types by means of equations and conditional equations.* Technical Report IW 131, Mathematical Centre, Dept. of Computer Science, Amsterdam, 1980. To appear in combined form with the previous reference in the *Journal of the Association for Computing Machinery*, 24 pp.

13 Bergstra, J. A., and Tucker, J. V. Algebraic specifications of computable and semicomputable data structures. 1983. To appear in *Theoretical Computer Science*, 24 pp.

14 Bergstra, J. A., Broy, M., Tucker, J. V. and Wirsing, M. On the power of algebraic specifications. In J. Gruska and M. Chytil (editors). *Mathematical foundations of computer science 1981*, Czechoslovakia, pages 192–204. Springer-Verlag, Berlin, 1980. Springer Lecture Notes in Computer Science, volume 118.

15 Birkhoff, G. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31, 433–54, 1935.

16 Birkhoff, G. and Lipson, J. Heterogeneous algebras. *Journal of Combinatorial Theory*, 8, 15–33, 1970.

17 Burstall, R. M. Proving properties of programs by structural induction. *Computer Journal*, 12(1), 41–8, 1969.

18 Burstall, R. M. and Goguen, J. A. Algebras, theories and freeness: an introduction for computer scientists. In *Proceedings, 1981 Marktoberdorf NATO Summer School*, Reidel, 1982.

19 Cartwright, R. A constructive alternative to axiomatic data type definitions. In *Proceedings of 1980 LISP Conference*, pages 46–55. Stanford University, 1980.

20 Cohn, P. M. *Universal algebra.* Harper and Row, 1965. Revised edition 1980.
21 Dahl, O.-J., Myhrhaug, B. and K. Nygaard. *The SIMULA 67 common base language.* Technical Report, Norwegian Computing Center, Oslo, 1970. Publication S-22.
22 Dershowitz, N. Ordering for term-rewriting systems. *Journal of Theoretical Computer Science,* 17(3), 279–301, 1982.
23 Dershowitz, N., Hsiang, J., Josephson, N. and Plaisted, D. *Associate-commutative rewriting.* Technical Report, University of Illinois, 1983. To appear in *IJCAI* 1983.
24 Frölich, A. and Shepherdson, J. C. Effective Procedures in Field Theory. *Philos. Trans. Roy. Soc. London ser. A,* 248, 407–32, 1956.
25 Ganzinger, H. *Parametric specifications, parameter passing and optimizing implementations.* Technical Report TUM-18110, Technical University of Munich, 1981.
26 Giarrantana, V., Gimona, F. and Montanari, U. Observability concepts in abstract data specifications. In *Proceedings, Conference on Mathematical Foundations of Computer Science,* Springer-Verlag, 1976.
27 Goguen, J. A. Some remarks on data structures. 1973. Abstract of Lectures at Eidgenössische Technische Hochschule, Zurich.
28 Goguen, J. A. Realization is universal. *Mathematical System Theory,* 6, 359–74, 1973.
29 Goguen, J. A. Semantics of computation. In *Proceedings, First International Symposium on Category Theory Applied to Computation and Control,* pages 234–49. University of Massachusetts at Amherst, 1974. Also published in Lecture Notes in Computer Science, Vol. 25, Springer-Verlag, 1975, pp. 151–63.
30 Goguen, J. A. Some design principles and theory for OBJ-0, a language for expressing and executing algebraic specifications of programs. In *Proceedings, International Conference on Mathematical Studies of Information Processing,* pages 429–75. IFIP Working Group 2.2, Kyoto, Japan, 1978.
31 Goguen, J. A. How to prove algebraic inductive hypotheses without induction: with applications to the correctness of data type representations. In W. Bibel and R. Kowalski (editors), *Proceedings, 5th Conference on Automated Deduction,* pages 356–73. Springer-Verlag, Lecture Notes in Computer Science, Volume 87, 1980.
32 Goguen, J. A. and Meseguer, J. Completeness of many-sorted equational logic. *SIGPLAN Notices,* 16(7), 24–32, July, 1981. Also appeared in *SIGPLAN Notices,* January 1982, 17, no. 1, pages 9–17; extended version as SRI Technical Report, 1982, and to be published in *Houston Journal of Mathematics.*
33 Goguen, J. A. and Meseguer, J. Universal realization, persistent interconnection and implementation of abstract modules. In *Proceedings, 9th International Colloquium on Automata, Languages and Programming,* Springer-Verlag, 1982. Lecture Notes in Computer Science, Volume 140.
34 Goguen, J. A. and Tardo, J. An introduction to OBJ: a language for writing and testing software specifications. In *Specification of reliable software,* pages 179–89. IEEE, 1979.
35 Goguen, J. A., Meseguer, J., and Plaisted, D. Programming with parameterized abstract objects in OBJ. In Ferrari, D., Bolognani, M. and Goguen, J. (editors), *Theory and practice of software technology,* pages 163–93. North-Holland, 1982.
36 Goguen, J. A., Thatcher, J. W. and Wagner, E. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh (editor), *Current trends in programming methodology,* pages 80–149. Prentice-Hall, 1978. Original version, IBM T. J. Watson Research Center Technical Report RC 6487, October 1976.

37 Goguen, J. A., Thatcher, J. W., Wagner, E. and Wright, J. B. Abstract data types as initial algebras and the correctness of data representations. In *Computer graphics, pattern recognition and data structure,* pages 89–93. IEEE, 1975.
38 Guttag, J. V. *The specification and application to programming of abstract data types.* Ph.D. thesis, University of Toronto, 1975. Computer Science Department, Report CSRG-59.
39 Higgins, P. J. Algebras with a scheme of operators. *Mathematische Nachrichten,* 27, 115–32, 1963.
40 Higman, G. Subgroups of finitely presented groups. *Proceedings of the Royal Society,* (A) 262, 455–75, 1961.
41 Hoare, C. A. R. Proof of correctness of data representation. *Acta Informatica,* 1, 271–81, 1972.
42 Huet, G. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the Association for Computing Machinery,* 27, 797–821, 1980. Preliminary version in 18th Symposium on Foundations of Computer Science, IEEE, 1977.
43 Huet, G. A complete proof of correctness of the Knuth and Bendix completion algorithm. *JCSS,* 23, 11–21, 1981.
44 Huet, G., and Hullot, J. M. Proofs by induction in equational theories with constructors. In *Symp. on Foundations of Computer Science,* IEEE, 1980.
45 Huet, G. and Oppen, D. Equations and rewrite rules: a survey. In Book, R. (editor), *Formal language theory: perspectives and open problems,* Academic Press, 1980.
46 Hullot, J. M. Canonical forms and unification. In W. Bibel and R. Kowalski (editors), *Proceedings, 5th Conference on Automated Deduction,* pages 318–34. Springer-Verlag, Lecture Notes in Computer Science, Volume 87, 1980.
47 Hupbach, U. L. Abstract implementation of abstract data types. In *MFCS '80,* pages 291–304. Springer-Verlag, 1980. Lecture Notes in Computer Science, volume 88.
48 Jackson, M. A. *Principles of program design.* Academic Press, 1975.
49 Jouannaud, J. P. Confluent and coherent equational term rewriting systems. In *Proc. 5th CAAP.* Springer Lecture Notes in Computer Science, 1983; to appear.
50 Jouannaud, J. P. and Kirchner, H. *Completion of a set of rules modulo a set of equations.* Technical Report, Centre de Recherche en Informatique de Nancy, 1983.
51 Jouannaud, J.-P., Kirchner, C., Kirchner, H. Incremental construction of unification algorithms in equational theories. In *Automata, Languages and Programming, Barcelona, 1983,* pages 361–73. Springer Lecture Notes in Computer Science, 154, 1983.
52 Jouannaud, J.-P., Lescanne, P., and Reinig, F. *Recursive decomposition ordering and multiset orderings.* Technical Report MIT/LCS/TM-219, MIT, Comp. Sci. Lab., 1982. To appear in 1982 IFIP T.C. 2.2, Garmisch-Partenkirchen, and in Inf. Proc. Letters.
53 Kaphengst, H. What is computable for abstract data types? In *Proceedings of FCT'81,* pages 173–81. Springer, 1981. Springer-Verlag Lecture Notes in Computer Science, volume 117.
54 Kirchner, H. *A general inductive completion algorithm and application to abstract data types.* Technical Report, Centre de Recherche en Informatique de Nancy, 1983.
55 Knuth, D. and Bendix, P. Simple word problems in universal algebra. In J. Leech (editor), *Computational problems in abstract algebra,* Pergamon Press, 1970.

56 Lankford, D. S. *Some remarks about inductionless induction.* Technical Report, Mathematics Department, Louisiana Tech University, 1980.

57 Lankford, D. S. *A simple explanation of inductionless induction.* Technical Report, Mathematics Department, Louisiana Tech University, 1981. MPT-14.

58 Lankford, D. and Ballantyne, A. *Decision procedures for simple equational theories with permutative axioms: complete sets of permutative reductions.* Technical Report, Univ. of Texas at Austin, Dept. of Mathematics and Computer Science, 1977. ATP-37.

59 Lawvere, F. W. Functorial semantics of algebraic theories. *Proceedings, National Academy of Sciences*, 50, 1963. Summary of Ph.D. Thesis, Columbia University.

60 Lawvere, F. W. An elementary theory of the category of sets. *Proceedings, National Academy of Sciences, U.S.A.*, 52, 1506–11, 1964.

61 Levitt, K., Robinson, L. and Silverberg, B. *The HDM Handbook.* Technical Report, SRI, International, Computer Science Lab, 1979. Volumes I, II, III.

62 Liskov, B. and Zilles, S. Specification Techniques for Data Abstraction. *IEEE Transactions on Software Engineering*, SE-1(1), 7–19, 1975.

63 Liskov, B. H., Moss, E., Schaffert, C., Scheifler, B., and Snyder, A. *CLU Reference Manual.* Technical Report, MIT, Lab for Computer Science, 1979.

64 MacLane, S. and Birkhoff, G. *Algebra.* Macmillan, 1967.

65 Majster, M. E. Limits of the algebraic specification of abstract data types. *SIGPLAN Notices*, 12, 37–42, Oct. 1977.

66 Majster, M. E. Data types, abstract data types and their specification problem. In *Theoretical computer science*, pages 89–127. North Holland Publishing Company, 1979.

67 Malcev, A. I. Constructive Algebras I. *Russian Mathematical Surveys*, 16(3), 77–129, 1961.

68 Malcev, A. I. *Algorithms and recursive functions.* Wolters-Noordhof Publishing Co., 1970.

69 Matijasevic, Y. V. Diophantine representation of recursively enumerable predicates. In *Proceedings, Second Scandinavian Logic Symposium*, pages 171–7. North-Holland, 1971.

70 Davis, M., Matijasevic, Y. V. and Robinson, J. Hilbert's tenth problem: diophantine equations: positive aspects of a negative solution. In *Mathematical developments arising from Hilbert problems*, pages 323–78. AMS, 1976. Proc. Symp. Pure Math., Volume 28.

71 Milner, R. *An algebraic definition of simulation between programs.* Technical Report CS-205, Stanford University, Computer Science Department, 1971.

72 Musser, D. On proving inductive properties of abstract data types. *Proceedings, 7th ACM Symposium on Principles of Programming Languages*, 1980.

73 Newman, M. H. A. On theories with a combinatorial definition of 'equivalence'. *Ann. Math.*, 43, 223–43, 1942.

74 Parnas, D. L. A technique for software module specification. *Communications of the Association for Computing Machinery*, 15, 1972.

75 Peterson, G. and Stickel, M. Complete sets of reductions for some equational theories. *JACM*, 28, 233–64, 1981.

76 Plaisted, D. *A recursively defined ordering for proving termination of term rewriting systems.* Technical Report R-78-943, University of Illinois, Computer Science Department, 1978.

77 Plotkin, G. Building-in equational theories. *Machine Intelligence*, 7, 73–90, 1972.

78 Rabin, M. Computable algebra: general theory and theory of computable fields. *Transactions of the American Mathematical Society*, 95, 341–60, 1960.

79 Reichel, H., Hupbach, U. R., and Kaphengst, H. *Initial algebraic specifications of data types, parameterized data types, and algorithms.* Technical Report, VEB Robotron, Zentrum fur Forschung und Technik, Dresden, 1980.

80 Rogers, H. *The theory of recursive functions and effective computability.* McGraw-Hill, 1967.

81 Stickel, M. A unification algorithm for associative-commutative functions. *JACM*, 28, 423–34, 1981.

82 Tarski, A. *Equational logic and equational theories of algebras.* North-Holland, 1968, pages 275–88.

83 Thatcher, J. W., Wagner, E. G. and Wright, J. B. Data type specification: parameterization and the power of specification techniques. In *Proceedings of 1979 POPL*, ACM, 1979.

84 Thatcher, J. W., Wagner, E. G., and Wright, J. B. Data type specification: parameterization and the power of specification techniques. *ACM TOPLAS*, 4(4), 711–32, Oct. 1982.

85 van der Waerden, B. L. *Moderne algebra, I (1st. ed.).* Julius Springer, 1930.

86 van der Waerden, B. L. Eine Bemerkung über die Unzerlebarkeit von Polynomen. *Math. Ann.*, 102, 738–9, 1930.

87 Van Wijngaarden, A. *et al.* Revised Report on the Algorithmic Language, ALGOL 68. *Acta Informatica*, 5, 1, 236, 1974.

88 Wand, M. *Algebraic theories and tree rewriting systems.* Technical Report 66, Computer Science Dept., Indiana University, 1977.

89 Wand, M. Final algebra semantics and data type extension. *J. Comp. Sys. Sciences*, 19, 27–44, 1979.

90 Warren, David. *An abstract prolog instruction set.* Technical Report Technical Note 309, SRI International, Artificial Intelligence Center, 1983.

91 Zilles, S. *Abstract specification of data types.* Technical Report 119, Computation Structures Group, MIT, 1974.

92 Zilles, S. An introduction to data algebras. 1975. Unpublished draft.