Theoretical Computer Science 43 (1986) 149-167
North-Holland

PARTIAL EVALUATION AND ω-COMPLETENESS OF ALGEBRAIC SPECIFICATIONS *

Jan HEERING

Centre for Mathematics and Computer Science, 1098 SJ Amsterdam, The Netherlands

Communicated by R. Milner
Received January 1985
Revised December 1985

Abstract. Suppose $P(x, y)$ is a program with two arguments, whose first argument has a known value c, but whose second argument is not yet known. *Partial evaluation* of $P(c, y)$ results (or rather: should result) in a specialized residual program $P_c(y)$ in which 'as much as possible' has been computed on the basis of c. In the literature on partial evaluation this is often more or less loosely expressed by saying that partial evaluation amounts to 'making maximal use of incomplete information'. In this paper a precise meaning is given to this notion in the context of equational logic, initial algebra specification, and term rewriting systems. If maximal propagation of incomplete information is to be achieved within this context, as a first step it is necessary to add equations to the algebraic specification in question until it is *ω-complete* (if ever). The basic properties of ω-complete specifications are discussed and some examples of ω-complete specifications as well as of specifications that do not have a finite ω-complete enrichment are given.

1. Introduction

1.1. Partial evaluation

The current investigation was inspired by the notion of *partial evaluation* or *mixed computation* as discussed, for instance, in [6] (which gives many references), and in [11, 12]. Although rather vague in scope, partial evaluation is basically a form of constant propagation. Suppose $P(x, y)$ is a program with two arguments, whose first argument has a known value c, but whose second argument is still unknown. Partial evaluation of $P(c, y)$ with unbound y results (or rather: should result) in a specialized residual program $P_c(y)$ in which 'as much as possible' has been computed on the basis of c. For instance, if P is a general context-free parser having as arguments a grammar and a string, partial evaluation of P with known grammar G and unknown string should lead to a specialized parser P_G by propagating G in P.

Partial evaluation is first and foremost an important unifying concept, shedding light on the relationship between interpretation and compilation, on the possible meaning of an ill-defined term like *compile-time*, on program optimization and program generators in general, and on type checking. Secondly, it is a useful technique in strictly limited and well-defined contexts in which the axioms and rules required can be hand-tailored to the application at hand.

* Partial support received from the European Communities under ESPRIT project 348 (Generation of Interactive Programming Environments—GIPE).

The notion of 'computing as much as possible on the basis of incomplete information' is widespread in the partial evaluation literature. As Ershov puts it [6, p. 49]: "A well-defined mixed computation which, in a sense, makes *a maximal use* of the information contained in the bound argument yields a rather efficient residual program." And Komorowski says [12, p. 59]: "Partial evaluation is a case of program transformation. It attempts to improve efficiency of program execution by eliminating run-time checks and *performing as much computation in advance as possible.* However, it does not modify algorithms." (Emphasis added in both cases.)

When experimenting with partial evaluation in the context of term rewriting systems [10], one quickly discovers that making a maximal use of incomplete information or computing as much in advance as possible is very difficult or even impossible. The rewrite rules used to evaluate *closed* (i.e., variable-free) terms are usually found to be inadequate when applied to *open* terms (i.e., terms containing variables) and numerous new and more general rules have to be added if anything like a canonical or in some sense simplest form is to be reached. Suppose, for example, that the following simple term-rewriting system R for a function max on the natural numbers with constant 0 and successor function S is given (with $1 = S(0)$):

$$\max(0, x) \to x, \qquad \max(x, 0) \to x,$$
$$\max(S(x), S(y)) \to S(\max(x, y)).$$

Partial evaluation of $\max(\max(1, 1), x)$ to $\max(1, x)$ requires no new rewrite rules, but for $\max(\max(1, x), 1)$ the same result can only be obtained by applying the commutative and associative properties of max, which are not needed for the evaluation of closed max-terms. Similarly, R is unable to reduce $\max(x, x)$ to x or $\max(S(x), x)$ to $S(x)$. In a larger context this implies that a term like

if $\max(x, x) = x$ **then** E **else** E' **fi**

cannot be reduced to E. This may block yet another reduction, and so on.

In general, the additional rewrite rules required correspond to valid equations from the viewpoint of initial algebra semantics [15]. In principle, new rules have to be added as long as the term rewriting system is incomplete with respect to the equational theory of the initial algebra in question. If, as a first step, one considers equations instead of rewrite rules, this means that new equations have to be added until the equational specification is complete with respect to the equational theory of the initial algebra (if ever), i.e., until the equational specification is *ω-complete.* As a second step one then has to consider the compilation of ω-complete specifications to term rewriting systems. The latter step falls outside the scope of this paper.

1.2. Algebraic specification, equational logic, and initial algebra semantics—some basic facts

In this section we shall give a brief summary of some basic facts of algebraic specification theory which are essential to an understanding of what follows. Good references are [2, 15].

An algebraic specification S consists of two parts:

(i) a many-sorted *signature* Σ_S, defining a language of strongly typed *terms* (expressions), and

(ii) a set E_S of *equations* (identities) between Σ_S-terms, defining an *equational theory* consisting of all equations provable from E_S by means of many-sorted *equational logic.*

The rules of inference of equational logic are essentially the rules of *reflexivity, symmetry, transitivity,* and *substitution.* Two more rules are needed if Σ_S has void sorts—see [15, Section 4.3].

Models of algebraic specifications are many-sorted algebras A such that (the interpretations of) all equations in E_S are valid in A. This is the well-known Tarski-semantics, but generalized to the many-sorted case.

If a Σ_S-equation is valid in *all* models of S, it is provable from E_S by means of equational logic. This is the *completeness property* of many-sorted equational logic. In general, however, one is not interested in the full class of models of an algebraic specification, but only in a single model (or isomorphism class of models) which is isomorphic to the algebra (the data type) one wishes to specify. The model closest to ordinary programming practice is the *initial algebra* I_S which is characterized by the following two properties:

(i) every element of I_S corresponds to at least one closed Σ_S-term ('no junk');

(ii) I_S is maximally free, which means that elements of I_S are never equal unless the corresponding closed terms can be proved equal from E_S ('no confusion').

Every algebraic specification (without void sorts) has an initial algebra which is uniquely determined up to isomorphism.

1.3. ω-Completeness of algebraic specifications

Because of the 'no junk' property, the initial algebra I_S of an algebraic specification S almost always has a much richer equational theory than can be derived from the equations E_S of S by means of equational logic alone, i.e., in general, equational logic is not complete with respect to the initial algebra. Although the closed equations valid in I_S can always be proved from E_S using equational reasoning, open equations valid in I_S do not in general yield to such simple means of deduction, but require stronger rules of inference (such as structural induction) for their proofs. For instance, consider the following specification:

```
module BOOL
begin
  sort bool
  functions F, T: →bool                    (false, true)
            ¬: bool→bool                   (not)
            +: bool×bool→bool              (exclusive-or)
            ., ∨: bool×bool→bool           (and, or)
```

equations $\neg F = T$
$\neg T = F$
$T + F = F + T = T$
$F + F = T + T = F$
$T.T = T$
$T.F = F.T = F.F = F$
$T \vee T = T \vee F = F \vee T = T$
$F \vee F = F$
end BOOL.

The initial model I_{BOOL} is a Boolean algebra with two elements. Because every closed term over Σ_{BOOL} is equal to T or F, proving the validity in I_{BOOL} of the laws of Boolean algebra (such as De Morgan's laws and the commutativity and associativity of $+$, $.$, and $\vee$) amounts to checking a finite number of closed instances for each law to be proved. These laws are not provable from E_{BOOL} by means of equational reasoning, however, as can be easily seen by constructing a model of BOOL in which they are false.

Completeness with respect to the equational theory of the initial algebra can be obtained in full generality by adding the so-called *ω-rule* to equational logic. This infinitary rule of inference allows one to infer an open Σ_S-equation e from a (possibly infinite) set of premises consisting of the closed Σ_S-instances of e. Using this extended version of equational logic, the equations valid in the initial algebra I_S can always be proved from E_S (even if they are not recursively enumerable!). Adding the ω-rule to equational logic has the general effect of making the class of models of a specification smaller and of highlighting the role of the initial model.

The ω-rule is rather unwieldy and the question arises whether it is possible to achieve completeness of a specification with respect to the equational theory of its initial algebra without transcending the limits of purely equational reasoning. More specifically, given a specification S, is it possible to add equations to it in such a way that (i) the initial algebra is not affected, and (ii) all open equations valid in the initial algebra become provable by purely equational means?

We shall call a specification having property (ii) *ω-complete.* We shall discuss the basic properties of nonparametrized ω-complete specifications (Section 2), give some examples (Section 3), and, finally, sketch an approach towards automatic addition of significant new equations valid in the initial algebra, i.e., automatic (partial) ω-enrichment (Section 4).

1.4. Related work

While revising this paper for publication, it was brought to my attention that the notion of ω-completeness as discussed in this paper was investigated by Paul [17] in the context of 'inductionless induction' under the name *inductive completeness.*[1]

[1] I am indebted to P. Lescanne for pointing this out to me.

Paul gives several examples of inductively complete algebraic specifications and their compilation to complete term rewriting systems (Sections 3.1–3.2 of this paper). He also shows that some specifications do not have a finite inductive closure, i.e., no finite ω-complete enrichment.

Taylor's survey [19] gives pointers to relevant work on (non)finitely based algebras done in the context of universal algebra, while in [5, 7] the equational theory of the natural numbers with addition, multiplication, and various other functions is discussed (Section 3.1 of this paper). Plotkin [18] has shown that the $\lambda K\beta\eta$-calculus is ω-incomplete (Section 3.4 of this paper).

Because the terminology in this field is rather confusing, a brief comparative list of terms used by various authors may be helpful:

- 'Inductive completeness' [17] = 'ω-completeness' (this paper),
- 'Inductive closure' [17] = 'ω-complete enrichment' (this paper),
- 'Inductive closure' [17] $\neq$ 'Inductive closure' [16],
- 'Inductive completion' [9] = 'Inductionless induction' [15, Section 6.7],
- 'Inductive completion' [9] $\neq$ 'Inductive closure' [17],
- 'Inductive completion' [9] $\neq$ 'Inductive closure' [16].

2. The ω-completeness property

Provable will always mean *provable by purely equational means* unless otherwise noted. Only finite specifications are considered. The semantics of a specification will always be the initial algebra semantics.

Definition 2.1. A finite algebraic specification S with signature Σ_S and set of Σ_S-equations E_S is *ω-complete* if every open equation all of whose closed Σ_S-instances are provable from E_S, is itself provable from E_S.

Theorem 2.2. *An algebraic specification S is ω-complete if and only if all equations valid in its initial algebra I_S are provable from E_S.*

Proof. For any S the closed equations valid in I_S are precisely the closed equations provable from E_S. Hence, the open equations valid in I_S are precisely the equations all of whose closed instances are provable from E_S. Hence, S is ω-complete if and only if not only every closed equation but also every open equation valid in I_S is provable from E_S. □

Theorem 2.3. *The equations valid in the initial algebra I_S of an ω-complete specification S are valid in all other models of S as well.*

Proof. According to Theorem 2.2, the equations valid in I_S are provable by purely equational means. Hence, according to the soundness property of equational logic, they are valid in all models of S. □

As explained in Section 1.3, open equations valid in the initial algebra of a specification generally require for their proofs rules of inference that are stronger than the simple rules of equational logic. Theorem 2.2 says that ω-complete specifications do not need these stronger rules of inference, i.e., they trade rules of inference for equational axioms. As far as their proofs are concerned, the open equations valid in the initial algebra of an ω-complete specification can be treated in the same way as their closed counterparts.

Theorem 2.4. *If an algebraic specification S is ω-complete, the set of equations valid in its initial algebra I_S is recursively enumerable.*

Proof. The set of equations valid in I_S is equal to the set of consequences of E_S according to Theorem 2.2. The latter set is recursively enumerable. □

Theorem 2.5. *If an algebraic specification S is ω-complete and if validity of closed equations in the initial algebra I_S is decidable, validity of open equations in I_S is decidable as well.*

Proof. On the one hand, the set of equations valid in I_S is recursively enumerable according to Theorem 2.4. On the other hand, each invalid open equation in I_S is finitely refutable, because the set of all of its closed instances is recursively enumerable and the validity of closed equations in I_S is decidable according to the second assumption of the theorem. □

Neither Theorem 2.4 nor Theorem 2.5 uses any specific properties of equational logic. In fact, their truth depends solely on the existence of a complete—but not necessarily purely equational—theory of the equations valid in the initial algebra.

Given a specification S, is there always a specification T such that

(i) $\Sigma_T = \Sigma_S$, $E_T \supseteq E_S$;

(ii) $I_T = I_S$;

(iii) T is ω-complete?

Even if I_S is finite, the answer is no. Lyndon has given an example of a single-sorted algebra with seven elements and one binary function, whose equational theory is not finitely based (not finitely axiomatizable) [14]. With this result he settled the question "Does every finite algebra possess a finite set of identities from which all others are derivable?" raised by him in [13]. As it has a (straightforward) initial algebra specification, this also means that Lyndon's algebra has no ω-complete initial algebra specification. Other examples are mentioned in [19, Section 9].

From an abstract data type viewpoint (but not necessarily from a strictly logical viewpoint), it is quite natural to allow extension of the signature with hidden sorts and functions. In that case ω-completeness can be achieved for a wider class of specifications. For instance, Lyndon's above-mentioned algebra has an ω-complete initial algebra specification with addition and multiplication mod 7 as hidden functions (see Section 3.2 for details).

Unlike the set of closed equations, the set of open equations valid in the initial algebra of a (finite) specification need not be recursively enumerable. For instance, the set of equations valid in the natural numbers with addition, multiplication, and a $<$-predicate is not recursively enumerable (see Section 3.1). Such an algebra cannot have an ω-complete specification according to Theorem 2.4. Extension of the signature does not help in such cases.

An obvious question is whether extension of the signature always helps if the equational theory of the initial algebra is recursively enumerable.

OPEN QUESTION 2.6. Suppose the set of equations valid in the initial algebra I_S of an algebraic specification S is recursively enumerable. Does this imply the existence of a specification T such that

(i) $\Sigma_T \supseteq \Sigma_S$, $E_T \supseteq E_S$;

(iia) T is conservative with respect to the closed theory of S, i.e., for all closed Σ_S-equations e,

$$E_T \vdash e \Rightarrow E_S \vdash e;$$

(iib) for every closed Σ_T-term t of a sort belonging to Σ_S there is a closed Σ_S-term t' such that

$$E_T \vdash t = t';$$

(iii) all equations valid in I_S are provable from E_T?

Note that T itself is not required to be ω-complete. This would be an even stronger requirement.

Consider a finitely generated algebra whose equational theory is recursively enumerable. The subset of closed equations valid in such an algebra is a fortiori recursively enumerable, and hence, according to [3, Theorem 4.1], it has a (finite) initial algebra specification with hidden sorts and functions. Hence, if the answer to Question 2.6 is affirmative, every finitely generated algebra with a recursively enumerable equational theory has an ω-complete initial algebra specification with hidden sorts and functions.

If the answer to Question 2.6 is affirmative, a further question is whether the hidden sorts can be dispensed with, that is, whether every specification has an ω-complete enrichment with hidden functions only. If the answer to this question is also affirmative, one would like to conclude that every finitely generated algebra with a recursively enumerable equational theory has an ω-complete initial algebra specification with hidden functions only. But this depends on yet another open problem: it is unknown whether every finitely generated algebra whose closed equational theory is recursively enumerable has an initial algebra specification with hidden functions only (see [4]).

3. Examples

This section contains two examples of nonparametrized ω-complete specifications (Sections 3.1–3.2), a discussion of the conditional function from the viewpoint of ω-completeness (Section 3.3), and a brief discussion of the ω-incompleteness of strong combinatory logic and related questions (Section 3.4).

3.1. The natural numbers with addition and multiplication

A simple initial algebra specification of the natural numbers with addition and multiplication looks as follows:

module NAT
begin
  sort N
  functions $0 :\rightarrow N$
  $S : N \rightarrow N$
  $+, . : N \times N \rightarrow N$
  variables $x, y :\rightarrow N$
  equations $x + 0 = x$ (1)
  $x + S(y) = S(x + y)$ (2)
  $x.0 = 0$ (3)
  $x.S(y) = x + (x.y)$ (4)
end NAT.

By adding the commutative, associative, and distributive laws for addition and multiplication, an ω-complete version of NAT is obtained:

module $\mathbb{N}$
begin
  include NAT
  variables $x, y, z :\rightarrow N$
  equations $x + y = y + x$ (5)
  $x + (y + z) = (x + y) + z$ (6)
  $x.y = y.x$ (7)
  $x.(y.z) = (x.y).z$ (8)
  $x.(y + z) = (x.y) + (x.z)$ (9)
end $\mathbb{N}$.

Theorem 3.1 ([7]). $\mathbb{N}$ *has the same initial algebra as* NAT *and is* ω*-complete.*

Sketch of proof. (a) $I_{\mathbb{N}} = I_{\text{NAT}}$, because (1) $\Sigma_{\mathbb{N}} = \Sigma_{\text{NAT}}$, and (2) the commutative, associative, and distributive laws for addition and multiplication are valid in I_{NAT} (proof by multiple structural induction).

(b) For every open or closed $\Sigma_{\mathbb{N}}$-term t, there is a $\Sigma_{\mathbb{N}}$-term P in canonical polynomial form such that $E_{\mathbb{N}} \vdash t = P$. Canonical forms are generated by the grammar

P::=0|**sum**,
sum::=**M**|(**sum**+**sum**),
M::=$S(0)$|**C**|**vars**|(**C**.**vars**),
vars::=**var**|(**vars**.**vars**),
var::=$x|y|\cdots$,
C::=$S(S(0))|S($**C**$)$,

with the additional condition that the number of monomials (maximal subterms produced by **M**) is minimal. Canonical forms are unique modulo associativity and commutativity of addition and multiplication. Two terms t_1 and t_2 are equal in $I_{\mathbb{N}}$ if and only if the corresponding canonical forms P_1 and P_2 are syntactically identical modulo the associative and commutative laws. Otherwise, there would be a nontrivial polynomial with *integer* coefficients which would be identically equal to zero. □

In [17] a proof of Theorem 3.1 is given based on a complete term rewriting system for $\mathbb{N}$. If *cut-off subtraction* $\dot{-}: N \times N \to N$ defined by the equations

$$x \dot{-} 0 = x,$$
$$0 \dot{-} x = 0,$$
$$S(x) \dot{-} S(y) = x \dot{-} y$$

is added to NAT, the equations valid in the initial algebra of the resulting specification NAT′ are not recursively enumerable [5, Section 8]. Hence, according to Theorem 2.4, no ω-complete specification of the natural numbers with addition, multiplication, and cut-off subtraction is possible. The same result holds if a $<$-predicate is added to NAT. (See also [17]. The same argument was used in [16] to show that equational reasoning plus structural induction is not necessarily complete with respect to the equational theory of the initial algebra.)

This shows that even in (seemingly) very simple cases complete partial evaluation is impossible.

3.2. Boolean algebra

BOOL of Section 1.3 is an ω-incomplete specification of Boolean algebra. An (almost) ω-complete version of BOOL is obtained by adding the equation $S(S(x)) = x$ to $\mathbb{N}$. This treatment of Boolean algebra is very economical and leads to an interesting canonical form for Boolean terms which is a direct descendant of the polynomial canonical form for $\Sigma_{\mathbb{N}}$-terms defined in the previous section. Consider the following module.

module $\mathbb{B}$
begin
  include $\mathbb{N}$ **with renaming** [$N \mapsto$ bool, $0 \mapsto F$, $S \mapsto \neg$]

functions $T: \rightarrow \text{bool}$
$\vee : \text{bool} \times \text{bool} \rightarrow \text{bool}$
variables $x, y : \rightarrow \text{bool}$
equations $\neg\neg x = x$ (10)
$x.x = x$ (11)
$T = \neg F$ (12)
$x \vee y = (x.y) + (x + y)$ (13)
end $\mathbb{B}$.

The successor function of $\mathbb{N}$ becomes negation in $\mathbb{B}$, addition becomes exclusive-or, multiplication becomes conjunction, etc. Equation (10) corresponds to $S(S(x)) = x$. Equation (11) has been added for the sake of ω-completeness.

Theorem 3.2. $\mathbb{B}$ *is an* ω*-complete specification of Boolean algebra.*

Proof. (a) $I_{\mathbb{B}} = I_{\text{BOOL}}$, because (1) $\Sigma_{\mathbb{B}} = \Sigma_{\text{BOOL}}$, (2) if $e \in E_{\text{BOOL}}$, then $E_{\mathbb{B}} \vdash e$ and hence, $I_{\mathbb{B}} \models e$, and (3) if $e \in E_{\mathbb{B}}$, then all closed $\Sigma_{\mathbb{B}}$-instances of e are provable from E_{BOOL} and hence, $I_{\text{BOOL}} \models e$.

(b) (See also part (b) of the proof of Theorem 3.1.) For every open or closed $\Sigma_{\mathbb{B}}$-term t there is a $\Sigma_{\mathbb{B}}$-term P in canonical form such that $E_{\mathbb{B}} \vdash t = P$. Canonical forms are generated by the grammar

P::= F | **sum**,
sum::=**M** | (**sum** + **sum**),
M::= T | **vars**,
vars::=**var** | (**vars**.**vars**),
var::= $x \mid y \mid \cdots$,

with the additional condition that the number of monomials is minimal and that all monomials are linear. Canonical forms are unique modulo the associative and commutative laws. Bringing a $\Sigma_{\mathbb{B}}$-term into canonical form involves the following steps (the equations of $\mathbb{N}$ **with renaming** $[N \mapsto \text{bool}, 0 \mapsto F, S \mapsto \neg]$ are numbered (1)-(9) in the same order in which they occur in $\mathbb{N}$):

Step 1. Eliminate all occurrences of $\vee$ and T by means of (13) and (12).

Step 2. Bring the resulting term into $\mathbb{N}$-canonical form (Section 3.1) (taking the renaming into account) by means of (1)-(9).

Step 3. (a) Reduce all coefficients to F or $\neg F$ by means of (10). Eliminate all coefficients of the form $\neg F$ by means of the equation $\neg F.x = x$ (which is provable from $E_{\mathbb{B}}$). Replace monomials consisting only of $\neg F$ by T by means of (12). Eliminate all monomials with coefficient F (except perhaps one) by means of (7), (3), (5), and (1).

(b) Linearize all monomials by means of (7), (8), and (11).

(c) Eliminate all monomials occurring more than once by means of (5)-(8), the equation $x+x=F$ (which is provable from $E_{\mathbb{B}}$), and (1).

Two terms t_1 and t_2 are equal in $I_{\mathbb{B}}$ if and only if the corresponding canonical forms P_1 and P_2 are syntactically identical modulo the associative and commutative laws. Otherwise, there would be a nontrivial P in canonical form such that $I_{\mathbb{B}} \models P = F$. But if P is of the form $\neg Q$, it assumes the value T because either Q is F or it assumes the value F if all variables have the value F. If P is not of the form $\neg Q$, consider a monomial q of P containing the least number of variables. Because monomials do not occur more than once, every other monomial contains at least one variable not occurring in q. If the variables occurring in q are given the value T and all other variables the value F, P assumes the value T. □

The canonical forms used in the above proof are Hsiang's 'normal expressions' [8]. Besides being the most natural ones from the present viewpoint, these canonical forms have the further merit of being the normal forms of a complete term-rewriting system which can be derived from $\mathbb{B}$ by a generalized Knuth-Bendix completion procedure. Other known canonical forms, such as the complete disjunctive normal form, do not have this property. Further details can be found in [8].

Paul [17] gives an ω-complete specification of the integers mod p with addition and multiplication (p prime) and proves Theorem 3.2 by taking $p=2$. (If p is not prime, ω-completeness is more difficult to achieve because the equation $x^p = x$, which corresponds to equation (11) of $\mathbb{B}$, no longer holds and the existence of zero-divisors gives rise to equations like $2x^2+2x=0 \pmod 4$ and $x^3+5x=0 \pmod 6$.) This result can be applied as follows. Consider Lyndon's example of a seven element algebra having no ω-complete initial algebra specification without hidden sorts and functions (Section 2). It has a straightforward initial algebra specification:

module L
begin
  sort A
  functions $0, 1, 2, 3, 4, 5, 6 : \rightarrow A$
  $\lambda : A \times A \rightarrow A$
  variable $x : \rightarrow A$
  equations $\lambda(4,1) = 4$
  $\lambda(4,2) = \lambda(5,1) = \lambda(5,2) = \lambda(5,3) = 5$
  $\lambda(4,3) = \lambda(6,1) = \lambda(6,2) = \lambda(6,3) = 6$
  $\lambda(0,x) = \lambda(1,x) = \lambda(2,x) = \lambda(3,x) = 0$
  $\lambda(x,0) = \lambda(x,4) = \lambda(x,5) = \lambda(x,6) = 0$
end L.

Every k-ary function on a set of p elements (p prime) corresponds to a polynomial in k variables over the integers mod p. Take $p=7$ and let $\mathbb{Z}_7$ be an ω-complete specification of the integers mod 7 with sort A, constants $0, \ldots, 6$, and functions +

and . , then L has the following ω-complete hidden function enrichment:

module $\mathbb{L}$
begin
  include $\mathbb{Z}_7$
  hidden functions $+, .$
  function $\lambda : A \times A \to A$
  variables $x, y : \to A$
  equation $\lambda(x, y) = 4.P_{4,1}(x, y)$
$+ 5.(P_{4,2}(x, y) + P_{5,1}(x, y) + P_{5,2}(x, y) + P_{5,3}(x, y))$
$+ 6.(P_{4,3}(x, y) + P_{6,1}(x, y) + P_{6,2}(x, y) + P_{6,3}(x, y))$
  where $P_{m,n}(x, y) = \prod_{i=0, i+m\neq 0}^{6} (x+i). \prod_{j=0, j+n\neq 0}^{6} (y+j)$
end $\mathbb{L}$.

$P_{m,n}(x, y)$ has the property

$$P_{m,n}(m, n) = 1 \quad \text{and} \quad P_{m,n}(x, y) = 0 \qquad \text{for } x \neq m,\ y \neq n.$$

The above method of obtaining an ω-complete hidden function enrichment applies to all single-sorted algebras with p elements (p prime).

3.3. The conditional function

The following module contains a simple definition of a polymorphic conditional function if:

module IF
begin
  include $\mathbb{B}$
  variable $\sigma : \to$ **sorts**
  function $\text{if} : \text{bool} \times \sigma \times \sigma \to \sigma$
  variables $u, v : \to \sigma$
  equations $\text{if}(F, u, v) = v$ (14)
$\text{if}(T, u, v) = u$ (15)
end IF.

Sort variable σ ranges over all sorts occurring in the specification, i.e., if IF is combined with a specification S, $\text{if} : \text{bool} \times \sigma \times \sigma \to \sigma$ expands into a nonpolymorphic $\text{if}_s : \text{bool} \times s \times s \to s$ for every sort $s \in \Sigma_{S+\text{IF}}$.

Let DIF be the union of IF and

module D
begin
  sort data
  functions $d_1, d_2, \ldots, d_m : \to \text{data} \quad (m > 1)$
end D.

In DIF the if-function has two nonpolymorphic instances, namely $\text{if}_{\text{bool}}: \text{bool} \times \text{bool} \times \text{bool} \to \text{bool}$ and $\text{if}_{\text{data}}: \text{bool} \times \text{data} \times \text{data} \to \text{data}$.

D is trivially ω-complete for $m > 1$, but in the degenerate case $m = 1$ the equation $u = d_1$ (with u a variable of sort data) is valid in I_D. From now on $m > 1$ is assumed.

DIF is not ω-complete. The equation

$$\text{if}(X, u, u) = u \qquad (*)$$

is an example of an equation which is valid in I_{DIF}, but not provable from E_{DIF}. In conventional programming languages, for instance, equations (14) and (15) hold but (*) does not, because the evaluation of X may loop or have side-effects.

The following version of IF is better from the viewpoint of ω-completeness:

module IFa
begin
  include IF
  variables $\sigma : \to$ **sorts**
  $u, v, w : \to \sigma$
  $X, Y, Z : \to \text{bool}$
  equations $\text{if}(X, u, v) = \text{if}(X, u, \text{if}(\neg X, v, w))$ (16)
  $\text{if}(X, u, \text{if}(Y, v, w)) = \text{if}(\neg X.Y, v, \text{if}(X, u, w))$ (17)
  $\text{if}(X, u, \text{if}(Y, u, v)) = \text{if}(X \vee Y, u, v)$ (18)
  $\text{if}(X, \text{if}(Y, u, v), w) = \text{if}(X.Y, u, \text{if}(X.\neg Y, v, w))$ (19)
  $\text{if}(X, Y, Z) = (X.Y) + (\neg X.Z)$ (20)
end IFa.

Theorem 3.3. DIFa = D + IFa *has the same initial algebra as* DIF *and is* ω*-complete.*

Proof. (a) $I_{\text{DIFa}} = I_{\text{DIF}}$, because $\Sigma_{\text{DIFa}} = \Sigma_{\text{DIF}}$ and all equations in E_{DIFa} are valid in I_{DIF}.

(b) If t is a Σ_{DIFa}-term of sort bool, it can be brought into $\mathbb{B}$-canonical form (Section 3.2) because all ifs can be eliminated from t by means of (20). If t is a Σ_{DIFa}-term of sort data containing distinct Boolean variables $X_1, \ldots, X_k$ $(k \geq 0)$ and distinct variables of sort data $u_1, \ldots, u_l$ $(l \geq 0)$, it can be brought into the canonical form δ_1 or

$$\text{if}(\xi_n, \delta_n, \text{if}(\xi_{n-1}, \delta_{n-1}, \ldots, \text{if}(\xi_1, \delta_1, v) \cdots)) \quad (n \geq 2).$$

The δ_i's are constants or variables of sort data (i.e., elements of $\{d_1, \ldots, d_m, u_1, \ldots, u_l\}$), v is an arbitrarily chosen variable of sort data, and the ξ_i's are Boolean terms in $\mathbb{B}$-canonical form such that

(i) $\delta_i \neq \delta_j$ $(i \neq j)$,
(ii) ξ_i is not of the form F or T,
(iii) $\xi_i.\xi_j =_{\mathbb{B}} F$ $(i \neq j)$,
(iv) $\bigvee_{i=1}^{n} \xi_i =_{\mathbb{B}} T$.

Two canonical forms are equal in I_{DIFa} if and only if they are syntactically identical modulo commutativity and associativity of . and +, modulo the shuffling of (ξ_i, δ_i)-pairs, and modulo the choice of v.

It takes the following steps to bring a Σ_{DIFa}-term of sort data into canonical form:

Step 1. Eliminate all Boolean ifs by means of (20).

Step 2. Eliminate all ifs from the second argument of other ifs by means of (19).

Step 3. Expand the innermost $\mathrm{if}(\xi, \delta, \delta')$ (if it exists) into $\mathrm{if}(\xi, \delta, \mathrm{if}(\neg\xi, \delta', v))$ by means of (16). The resulting term satisfies (iv).

Step 4. Merge all ifs whose second argument contains the same constant or variable by means of (17) and (18). The resulting term satisfies (i).

Step 5. If at this point the canonical form in statu nascendi is of the form

$$\mathrm{if}(\eta_n, \delta_n, \mathrm{if}(\eta_{n-1}, \delta_{n-1}, \ldots, \mathrm{if}(\eta_1, \delta_1, v) \cdots)) \quad (n > 1),$$

then turn it inside out, i.e., turn it by means of $\frac{1}{2}n(n-1)$ applications of (17) into

$$\mathrm{if}(\theta_1, \delta_1, \ldots, \mathrm{if}(\theta_{n-1}, \delta_{n-1}, \mathrm{if}(\theta_n, \delta_n, v)) \cdots),$$

with $\theta_n = \eta_n$, $\theta_{n-1} = \neg\eta_n.\eta_{n-1}$, $\theta_{n-2} = \neg(\neg\eta_n.\eta_{n-1}).(\neg\eta_n.\eta_{n-2})$, etc. The resulting term satisfies (iii).

Step 6. Bring all θ_i's into $\mathbb{B}$-canonical form ξ_i.

Step 7. (a) If ξ_i is of the form F for some i, eliminate the corresponding if and δ_i by means of (14).

(b) If ξ_i is of the form T for some i, the term is of the form $\mathrm{if}(T, \delta, \nu)$ because of property (iii) and Step 7(a). Reduce it to δ by means of (15). The resulting term satisfies (ii) and is in canonical form. □

Although, according to Theorem 3.3, IFa is ω-complete when combined with the simplest possible D, ω-completeness is lost if D is somewhat more complicated. For instance, the equations

$$S(\mathrm{if}(X, x, y)) = \mathrm{if}(X, S(x), S(y)),$$

$$\mathrm{if}(X, x, y).\mathrm{if}(X, y, x) = x.y$$

are valid in $I_{\mathbb{N}+\mathrm{IFa}}$ but not provable from $E_{\mathbb{N}+\mathrm{IFa}}$. This can be remedied by adding the distributive property of if to IFa:

module IFb
begin
  include IFa
  variables $X :\to bool$
  $\sigma, \tau :\to$ **sorts**
  $u, v :\to \sigma$
  $\Phi : \sigma \to \tau$
  equation $\Phi(\mathrm{if}(X, u, v)) = \mathrm{if}(X, \Phi(u), \Phi(v))$ (21)
end IFb.

Equation (21) is to be interpreted as follows. If IFb is combined with a specification S, (21) expands into n separate instances for every n-ary function $f \in \Sigma_{S+\mathrm{IFb}}$ by substitution of $(\lambda x_k) f(x_1, \ldots, x_k, \ldots, x_n)$ for Φ $(1 \leqslant k \leqslant n)$. For example, one of the instances of (21) is $(f = \mathrm{if}, k = 2)$

$$\mathrm{if}(Y, \mathrm{if}(X, u, v), w) = \mathrm{if}(X, \mathrm{if}(Y, u, w), \mathrm{if}(Y, v, w)),$$

which is provable from E_{IFa}.

Theorem 3.4. *$S + \mathrm{IFb}$ is ω-complete for every ω-complete specification S that does not contain functions of one or more Boolean arguments or with a Boolean result.*

Proof. Use for every sort $s \in \Sigma_S$ a canonical form similar to the one used in the proof of Theorem 3.3, but with δ_i a term of sort s in S-canonical form. To bring a term into canonical form, follow Steps 1-7 of Theorem 3.3 with two additional steps between Steps 1 and 2, and a slightly different Step 4':

Step 1.1. Move all ifs to outermost positions by means of (21).

Step 1.2. Bring all maximal if-free subterms (all of which are necessarily of the same sort) into S-canonical form.

Step 4'. Merge all ifs whose second argument contains syntactically identical S-canonical forms by means of (17) and (18). The resulting term satisfies (i). □

If S contains functions of Boolean arguments or with a Boolean result (as indeed it will in all realistic cases), the selective action of the first argument of the if-function gives rise to new equations and Theorem 3.4 fails. For instance, suppose an ω-complete specification S containing $\mathbb{B}$ is sufficiently-complete with respect to $\mathbb{B}$, i.e., all closed Σ_S-terms of sort bool can be proved equal to T or F. Suppose further that Σ_S contains a sort data and functions f, g : bool → data and h, k : bool × bool → data. In that case some typical equations valid in $I_{S+\mathrm{IFb}}$ but not provable from $E_{S+\mathrm{IFb}}$ are

$$\mathrm{if}(X, f(X), g(X)) = \mathrm{if}(X, f(T), g(F)), \tag{22}$$

$$\mathrm{if}(X + Y, h(X, Y), k(X, Y)) = \mathrm{if}(X + Y, h(X, \neg X), k(X, X)), \tag{23}$$

$$\mathrm{if}(X.Y, h(X, Y), k(X, Y)) = \mathrm{if}(X.Y, h(T, T), \mathrm{if}(X + Y, k(X, \neg X), k(F, F))). \tag{24}$$

Contrary to equations (14)-(21), which are valid in $I_{S+\mathrm{IF}}$ for all S satisfying the sufficient-completeness requirement just mentioned, equations like (22)-(24) are very much dependent on the particular S involved.

If interpreted as a left-to-right rewrite rule, equation (24) is typical of a whole class of rules whose right-hand sides contain more ifs than their left-hand sides. Application of such rules easily leads to terms containing an enormous number of alternatives, because, in general, most of the new branches only lead to further branches.

3.4. Combinatory logic

Consider the following algebraic specification of strong combinatory logic:

module CLX
begin
  sort F
  functions $\mathrm{K}, \mathrm{S} :\to F$
  $. : F \times F \to F$ (application)
  Note. The infix dot is not written and application associates to the left, i.e., $(\mathrm{K}.x).y$ is written as $\mathrm{K}xy$, etc.
  variables $x, y, z :\to F$
  equations $\mathrm{K}xy = x$
  $\mathrm{S}xyz = xz(yz)$
  $\mathrm{S}(\mathrm{S}(\mathrm{KS})(\mathrm{S}(\mathrm{KK})(\mathrm{S}(\mathrm{KS})\mathrm{K})))(\mathrm{KK}) = \mathrm{S}(\mathrm{KK})$
  $\mathrm{S}(\mathrm{KS})(\mathrm{S}(\mathrm{KK}))$
  $= \mathrm{S}(\mathrm{KK})(\mathrm{S}(\mathrm{S}(\mathrm{KS})(\mathrm{S}(\mathrm{KK})(\mathrm{SKK})))(\mathrm{K}(\mathrm{SKK})))$
  $\mathrm{S}(\mathrm{K}(\mathrm{S}(\mathrm{KS})))(\mathrm{S}(\mathrm{KS})(\mathrm{S}(\mathrm{KS})))$
  $= \mathrm{S}(\mathrm{S}(\mathrm{KS})(\mathrm{S}(\mathrm{KK})(\mathrm{S}(\mathrm{KS})(\mathrm{S}(\mathrm{K}(\mathrm{S}(\mathrm{KS})))\mathrm{S}))))(\mathrm{KS})$
  $\mathrm{S}(\mathrm{S}(\mathrm{KS})\mathrm{K})(\mathrm{K}(\mathrm{SKK})) = \mathrm{SKK}$
end CLX.

CLX is identical to $\mathbf{CL}+\mathbf{A}_{\beta\eta}$ in [1]. Hence, according to [1, Theorem 7.3.14], CLX is equivalent to the $\lambda K\beta\eta$-calculus. The last four closed equations (the so-called *combinatory axioms*) give CLX the *extensional property*, i.e., if, for two (possibly open) Σ_{CLX}-terms f and g not containing the variable x,

$$E_{\mathrm{CLX}} \vdash fx = gx,$$

then also

$$E_{\mathrm{CLX}} \vdash f = g.$$

Is CLX *ω-extensional*? That is, does

$$E_{\mathrm{CLX}} \vdash fa = ga \quad \text{for all closed } a$$

imply

$$E_{\mathrm{CLX}} \vdash f = g?$$

Plotkin has shown that the $\lambda K\beta\eta$-calculus is not ω-extensional [18; 1, Theorem 17.3.30]. Hence, CLX is not ω-extensional either. As

$$\omega\text{-completeness} + \text{extensionality} \Rightarrow \omega\text{-extensionality}, \tag{25}$$

CLX is not ω-complete. In fact, as far as CLX is concerned the notions of ω-extensionality and ω-completeness are equivalent. This is not difficult to prove. In view of (25) plus the fact that CLX is combinatorially complete, it is enough to show that

$$\text{combinatorial completeness} + \omega\text{-extensionality} \Rightarrow \omega\text{-completeness}. \tag{26}$$

Consider a Σ_{CLX}-equation $f = g$ all of whose closed instances are provable from E_{CLX}. Assume further that f and g contain the same variables $x_1, \ldots, x_k$ $(k \geqslant 1)$. (If f contains a variable x not in g, then replace some variable or constant v in g by Kvx, etc.) By combinatorial completeness of CLX, there exist closed terms ϕ and ψ such that

$$E_{CLX} \vdash f = \phi x_1 \ldots x_k,\ g = \psi x_1 \ldots x_k.$$

Applying ω-extensionality k times gives

$$E_{CLX} \vdash \phi = \psi.$$

Hence,

$$E_{CLX} \vdash \phi x_1 \ldots x_k = \psi x_1 \ldots x_k$$

and

$$E_{CLX} \vdash f = g.$$

This proves (26).

Two questions we have not succeeded in answering are the following.

OPEN QUESTION 3.5. Are the open equations valid in the initial algebra of CLX recursively enumerable?

OPEN QUESTION 3.6. Does CLX have an ω-complete enrichment in the sense of Question 2.6?

If—as would be our guess—the answer to the first question is no, the answer to the second question must also be no according to Theorem 2.4. If the answer to the first question is yes, the second question is a special case of Question 2.6.

4. Towards automatic (partial) ω-enrichment

Describing semantics by means of term rewriting systems has the advantage of yielding evaluators that work on both closed and open terms. Their performance on open terms (partial evaluation) is often disappointing, however, as many more or less trivial simplifications of open terms are beyond the power of the rewrite rules required for evaluating closed terms (Section 1.1). In such cases even rudimentary ω-enrichment may be rewarding, and the question arises whether this can be done automatically. (Even if the answer to Question 2.6 is affirmative, partial ω-enrichment is the best one can hope for in many cases. See Section 3.1.)

While 'inductionless induction' or 'inductive completion' algorithms (Section 1.4) can sometimes help in proving the validity of a given potential ω-enrichment, they do not help in suggesting significant new ω-enrichments (or, for that matter, in giving ω-completeness proofs).

An approach we are currently investigating is automatic partial ω-enrichment by means of sets of *enrichment rules*. This roughly works as follows. An enrichment rule

$$P(\sigma_1, \ldots, \sigma_m, \Phi_1, \ldots, \Phi_n) \rightarrow E(\sigma_1, \ldots, \sigma_m, \Phi_1, \ldots, \Phi_n)$$

is a *specifiction rewrite rule* consisting of a *specification pattern* P and an *enrichment scheme* E. The signatures of P and E contain *sort variables* σ_i and *function variables* Φ_j. If P matches the specification to be enriched S, i.e., if there is an instance of P which is a subspecification of S, the part of S matched by P is replaced by the corresponding instance of the enrichment scheme E, possibly after renaming the hidden sorts and functions introduced by E to avoid name clashes with the hidden items of S. Special care has to be taken to ensure that enrichment steps are correct.

This approach has the advantage of being rather natural. Its success depends on whether a large enough number of generally applicable enrichment rules can be found and on whether the validity of enrichment steps can be guaranteed.

Acknowledgment

While writing this paper we had helpful discussions with Jan Bergstra, Paul Klint, Jan Willem Klop, and Ed Kuijpers.

References

[1] H.P. Barendregt, *The Lambda Calculus* (North-Holland, Amsterdam, 1981).

[2] R.M. Burstall and J.A. Goguen, Algebras, theories and freeness: An introduction for computer scientists, in: M. Broy and G. Schmidt, eds., *Theoretical Foundations of Programming Methodology* (D. Reidel, Boston MA/Dordrecht, 1982) 329-348.

[3] J.A. Bergstra and J.V. Tucker, Algebraic specifications of computable and semi-computable data structures, Rept. IW 115/79, Dept. Computer Science, Centre for Mathematics and Computer Science, Amsterdam, 1979.

[4] J.A. Bergstra and J.V. Tucker, Initial and final algebra semantics for data type specifications: Two characterization theorems, *SIAM J. Comput.* **12**(2) (1983) 366-387.

[5] M. Davis, Y. Matijasevic and J. Robinson, Hilbert's tenth problem: Positive aspects of a negative solution, in: F.E. Browder, ed., *Mathematical Developments Arising from Hilbert Problems* (American Mathematical Society, Providence, RI, 1976) 323-378.

[6] A.P. Ershov, Mixed computation: Potential applications and problems for study, *Theoret. Comput. Sci.* **18** (1982) 41-67.

[7] L. Henkin, The logic of equality, *Amer. Math. Monthly* **84** (1977) 597-612.

[8] J. Hsiang, Topics in automated theorem proving and program generation, Rept. UIUCDCS-R-82-1113, Dept. Computer Science, Univ. Illinois at Urbana-Champaign, 1982.

[9] G. Huet, G. and J.M. Hullot, Proofs by induction in equational theories with constructors, *J. Comput. System Sci.* **25** (1982) 239-266.

[10] G. Huet and D.C. Oppen, Equations and rewrite rules: A survey, in: R.V. Book, ed., *Formal Languages: Perspectives and Open Problems* (Academic Press, New York/London, 1980).

[11] N.D. Jones, P. Sestoft and H. Søndergaard, An experiment in partial evaluation: The generation of a compiler generator, Rept. 85/1, Institute of Datalogy, Univ. Copenhagen, 1985.

[12] H.J. Komorowski, A Specification of an Abstract PROLOG Machine and its Application to Partial Evaluation, Dissertation No. 69, Linköping University, 1981.

[13] R.C. Lyndon, Identities in two-valued calculi, *Trans. Amer. Math. Soc.* **71** (1951) 457–465.
[14] R.C. Lyndon, Identities in finite algebras, *Proc. Amer. Math. Soc.* **5** (1954) 8–9.
[15] J. Meseguer and J.A. Goguen, Initiality, induction, and computability, in: M. Nivat and J. Reynolds, eds., *Algebraic Methods in Semantics* (Cambridge University Press, London, 1986).
[16] F. Nourani, On induction for programming logic: Syntax, semantics, and inductive closure, *Bull. European Assoc. Theoret. Comput. Sci. 13* (February 1981) 51–64.
[17] E. Paul, Proof by induction in equational theories with relations between constructors, in: B. Courcelle, ed., *9th Coll. on Trees in Algebra and Programming* (Cambridge University Press, London, 1984).
[18] G.D. Plotkin, The λ-calculus is ω-incomplete, *J. Symbolic Logic* **39** (1974) 313–317.
[19] W. Taylor, Equational logic, *Houston J. Math.*, Survey 1979.