Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J. Heering

Partial evaluation and ω-completeness of algebraic specifications

Department of Computer Science Report CS-R8501 January

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

PARTIAL EVALUATION AND

ω-COMPLETENESS OF ALGEBRAIC SPECIFICATIONS

Jan Heering
Centre for Mathematics and Computer Science
Amsterdam

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 initial algebra specifications and term rewriting systems. It turns out that, if maximal propagation of incomplete information is to be achieved, 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.

1983-84 CR Categories: D.1.2 [**Programming Techniques**]: Automatic Programming; D.3.4 [**Programming Languages**]: Processors - *Optimization;* F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages - *Algebraic Approaches to Semantics;* F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic - *Lambda-Calculus and Related Systems;* I.1.1 [**Algebraic Manipulation**]: Expressions and their Representation - *Simplification of Expressions;* I.2.2 [**Artificial Intelligence**]: Automatic Programming.

1980 Mathematics Subject Classification: 03B40 [**General Logic**]: Combinatory Logic and Lambda-Calculus; 03C05 [**Model Theory**]: Universal Algebra; 68B10 [**Software**]: Analysis of Programs - *Semantics;* 68C20 [**Metatheory**]: Symbolic Computation.

Key Words & Phrases: algebraic specification, ω-completeness, initial algebra semantics, equational logic, structural induction, ω-rule, program transformation, program optimization, partial evaluation, mixed computation, symbolic computation, propagation of incomplete information, constant propagation, combinatory logic, ω-extensionality.

Note: Partial support received from the European Communities under ESPRIT project no. 348 (Generation of Interactive Programming Environments).

Note: This paper has been submitted for publication elsewhere.

1. Introduction

1.1. Partial evaluation

The current investigation was inspired by the notion of *partial evaluation* or *mixed computation* as discussed for instance by Futamura [1], Beckman *et al.* [2], Ershov [3], and Komorowski [4,5]. 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 $P(c,y)$ 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.

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

Report CS-R8501
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

compile-time, on program optimization 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.

The notion of "computing as much as possible on the basis of incomplete information" is widespread in the partial evaluation literature. As Ershov - rather optimistically - puts it ([3], 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 ([4], 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 (Huet & Oppen [6]), one quickly discovers that making 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 satisfactory result is to be achieved. Suppose, for example, that the following rewrite rules for a function *max* on the natural numbers with constant 0 and successor function S are given (with $1 = S(0)$):

$$
\begin{aligned}
&max(0,x) \rightarrow x\\
&max(x,0) \rightarrow x\\
&max(S(x),S(y)) \rightarrow S(max(x,y)).
\end{aligned}
$$

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 required for the evaluation of closed *max*-terms.

Very often, the additional rewrite rules required correspond to valid equations from the viewpoint of initial algebra semantics (Goguen & Meseguer [7]). 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 algebraic specification is complete with respect to the equational theory of the initial algebra (if ever), i.e. until the equational specification is ω-complete. (The validity of such new equations can sometimes be checked by means of an inductive completion algorithm. See for instance Huet & Hullot [8].) As a second step one then has to consider the compilation of ω-complete specifications to term rewriting systems. Although this latter step is touched upon in some of the examples, it is not the primary topic of this paper.

1.2. ω-completeness of algebraic specifications

Consider a finite algebraic specification S with signature Σ and set of Σ-equations E. If a Σ-equation is valid in *all* models of S, it is provable from E by purely equational reasoning. This is the completeness property of many-sorted equational logic (Goguen & Meseguer [7]).

The completeness property does not in general extend to the equational theory of the initial algebra I of S. Although the closed equations valid in I can always be proved from E using equational reasoning, open equations valid in I 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 $F, T: \to bool$ (false, true)
    $\neg: bool \to bool$ (not)
    $+: bool \times bool \to bool$ (exclusive-or)
    $.\,, \vee: bool \times bool \to bool$ (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 easily be 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-equation e from a (possibly infinite) set of premises consisting of the closed Σ-instances of e. Using this extended version of equational logic, the equations valid in the initial algebra of a specification 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?

I shall call a specification having property (ii) *ω-complete*. I shall discuss the basic properties of non-parameterized ω-complete specifications (§2) and give some examples (§3).

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 2.1: 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 2.2: 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 2.1, the equations valid in I_S are provable by purely equational means. Hence, according to the completeness property of equational logic they are valid in all models of S. □

Theorem 2.3: For a given specification S and any Σ_S-equation e

$$I_S \models e$$

if and only if for all closed Σ_S-equations $t_1=t_2$

$$E_S \cup \{e\} \vdash t_1=t_2 \Rightarrow E_S \vdash t_1=t_2.$$

Proof: ($\Rightarrow$) $I_S \models e$ implies that all closed instances of e are provable from E_S.
($\Leftarrow$) All closed instances of e are provable from $E \cup \{e\}$, and hence, according to the assumption, from E. Hence $I_S \models e$. □

As explained in §1.2, 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 2.1 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 2.1. 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 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 (using a somewhat different terminology) has given an example of an algebra with one sort, seven elements, and one binary function, which has a straightforward initial algebra specification but no ω-complete initial algebra specification [9]. Other examples (also described in somewhat different terms) can be found in §67 of Grätzer [10].

If extension of the signature with auxiliary (hidden) sorts and functions is allowed, ω-completeness can be achieved for a wider class of specifications. The binary function in Lyndon's above-mentioned seven element algebra, for instance, can (like any other binary function on a set of seven elements) be expressed as a polynomial in two variables with coefficients in $\mathbb{Z}_7$, the integers mod 7. $\mathbb{Z}_7$ with addition and multiplication has an ω-complete specification very similar to the ω-complete specification of the Booleans discussed in §3.2.

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 §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 2.1: 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) $\Sigma_T \supseteq \Sigma_S$, $E_T \supseteq E_S$;
(ii*a*) T is conservative with respect to the closed theory of S, i.e. for all closed Σ_S-equations $t_1 = t_2$

$$E_T \vdash t_1 = t_2 \Rightarrow E_S \vdash t_1 = t_2;$$

(ii*b*) 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) T is ω-complete.

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 theorem 4.1 of Bergstra & Tucker [11], it has a (finite) initial algebra specification with auxiliary sorts and functions. Hence, if the answer to question 2.1 is affirmative, every finitely generated algebra with a recursively enumerable equational theory has an ω-complete initial algebra specification with auxiliary sorts and functions.

If the answer to question 2.1 is affirmative, a further question is whether the auxiliary sorts can be dispensed with. 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 auxiliary 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 auxiliary functions only (see Bergstra & Tucker [12]).

3. EXAMPLES

This section contains two examples of non-parameterized ω-complete specifications (§§3.1-2), a discussion of the conditional function from the viewpoint of ω-completeness (§3.3), and a brief discussion of the ω-incompleteness of strong combinatory logic and related questions (§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 $0: \to N$
    $S: N \to N$
    $+, . : N \times N \to N$
  variables $x, y : \to N$
  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 $x, y, z : \to N$
  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 3.1.1: $\mathbb{N}$ has the same initial algebra as NAT and is ω-complete.

PROOF: (*a*) $I_{\mathbb{N}} = I_{NAT}$, because (1) $\Sigma_{\mathbb{N}} = \Sigma_{NAT}$, and (2) the commutative, associative and distributive laws for addition and multiplication are valid in I_{NAT} (proof by multiple structural induction).
(*b*) $E_{\mathbb{N}} \vdash t = P$ for every open or closed $\Sigma_{\mathbb{N}}$-term t, where P is a term in *canonical form* generated by the grammar

P ::= 0 | **sum** | S(**P**)
sum ::= **product** | (**sum** + **sum**)
product ::= **variable** | (**product** . **product**)
variable ::= x | y | $\cdots$.

Canonical forms are unique *modulo* associativity and commutativity of addition and multiplication. Consider the following term rewriting system $R_{\mathbb{N}}$:

$$
\begin{aligned}
&x+0 \rightarrow x\\
&0+x \rightarrow x\\
&x+S(y) \rightarrow S(x+y)\\
&S(x)+y \rightarrow S(x+y)\\
&x.0 \rightarrow 0\\
&0.x \rightarrow 0\\
&x.S(y) \rightarrow x+(x.y)\\
&S(x).y \rightarrow y+(x.y)\\
&x.(y+z) \rightarrow (x.y)+(x.z)\\
&(x+y).z \rightarrow (x.z)+(y.z)\,.
\end{aligned}
$$

$R_{\mathbb{N}}$ is strongly terminating (use a recursive path ordering [6] with $. \succ + \succ S$) and confluent *modulo* the associative and commutative laws (apply theorem 3.3 of Huet [13]). $E_{\mathbb{N}} \vdash t_1=t_2$ for all rules $t_1 \rightarrow t_2 \in R_{\mathbb{N}}$. Furthermore, $R_{\mathbb{N}}$ reduces the left- and right-hand sides of all equations $t_1=t_2 \in E_{\mathbb{N}}$ to normal forms that are syntactically identical *modulo* the associative and commutative laws. Hence $R_{\mathbb{N}}$ is complete. The normal forms of $R_{\mathbb{N}}$ are precisely the canonical forms defined above.
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 non-trivial polynomial in one variable with *integer* coefficients having an infinity of zeros. □

If *cut-off subtraction* $\dot{-} : N \times N \rightarrow N$ defined by the equations

$$
\begin{aligned}
&x \dot{-} 0 = x\\
&0 \dot{-} x = 0\\
&S(x) \dot{-} S(y) = x \dot{-} y
\end{aligned}
$$

is added to NAT, the equations valid in the initial algebra of the resulting specification NAT' are not recursively enumerable (see §8 of Davis *et al.* [14]). Hence, according to proposition 2.4 no ω-complete specification of the natural numbers with addition, multiplication and cut-off subtraction is possible. A similar result holds if a $<$-predicate is added to NAT.

3.2. Boolean algebra

$BOOL$ of §1.2 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 canonical form for $\Sigma_{\mathbb{N}}$-terms defined in the previous paragraph. Consider

module $\mathbb{B}$
begin
  include $\mathbb{N}$ **with renaming** $[N \mapsto bool,\ 0 \mapsto F,\ S \mapsto \neg]$
  functions $T: \rightarrow bool$
    $\vee: bool \times bool \rightarrow bool$
  variables $x, y: \rightarrow bool$
  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 inversion 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 3.2.1: $\mathbb{B}$ is an ω-complete specification of Boolean algebra.

PROOF: (*a*) $I_{\mathbb{B}}=I_{BOOL}$, because (1) $\Sigma_{\mathbb{B}}=\Sigma_{BOOL}$, (2) if $e \in E_{BOOL}$, 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 $I_{BOOL} \models e$.
(*b*) (See also part (*b*) of the proof of theorem 3.1.1.) $E_{\mathbb{B}} \vdash t=P$ for every open or closed $\Sigma_{\mathbb{B}}$-term t, where P is a term in *canonical form* generated by the grammar

$$\begin{aligned}
&\mathbf{P} ::= F \mid \neg F \mid \mathbf{sum} \mid \neg\mathbf{sum}\\
&\mathbf{sum} ::= \mathbf{product} \mid (\mathbf{sum} + \mathbf{sum})\\
&\mathbf{product} ::= \mathbf{variable} \mid (\mathbf{product}\,.\,\mathbf{product})\\
&\mathbf{variable} ::= x \mid y \mid \cdots ,
\end{aligned}$$

and such that no two maximal multiplicative subterms are identical *modulo* commutativity and associativity of . and no multiplicative subterm contains the same variable more than once. 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** $[N \mapsto bool, 0 \mapsto F, S \mapsto \neg]$ are numbered (1)-(9) in the same order in which they occur in $\mathbb{N}$):

(S1a) Eliminate all occurrences of $\vee$ by means of (13).

(S1b) Eliminate all occurrences of T by means of (12).

(S2) Bring the resulting term into $\mathbb{N}$-canonical form (§3.1) (taking the renaming into account) by means of (1)-(9).

(S3a) Eliminate multiple occurrences of $\neg$ from the head of the resulting term by means of (10).

(S3b) Linearize all multiplicative subterms by means of (7), (8) and (11).

(S3c) Eliminate all maximal multiplicative subterms 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 non-trivial 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 maximal multiplicative subterm q of P containing the least number of variables. Because maximal multiplicative subterms do not occur more than once, every other maximal multiplicative subterm 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 virtually identical to the "normal expressions" of Hsiang [15]. 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 (similar to $R_{\mathbb{N}}$) 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 [15].

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 $\sigma: \rightarrow$ **sorts**
function $if: bool \times \sigma \times \sigma \rightarrow \sigma$
variables $u, v: \rightarrow \sigma$
equations $if(F, u, v) = v$ (1)
$if(\neg F, u, v) = u$ (2)
end *IF*.

If *IF* is combined with a specification S, $if: bool \times \sigma \times \sigma \rightarrow \sigma$ expands into a non-polymorphic $if_s: bool \times s \times s \rightarrow s$ for every sort $s \in \Sigma_{S \cup IF}$. Let DIF be the union of *IF* and

module *D*
begin
sort *data*
functions $d_1, d_2, \ldots, d_m: \rightarrow data$ $(m > 1)$
end *D*.

In *DIF* the *if*-function has two non-polymorphic instances, namely $if_{bool}: bool \times bool \times bool \rightarrow bool$ and $if_{data}: bool \times data \times data \rightarrow data$.

D is trivially ω-complete for $m > 1$, but in the degenerate case $m = 1$ the equation $u = d_1$ is valid in I_D. From now on $m > 1$ is assumed. *DIF* is not ω-complete. The equation $if(X, u, u) = u$ is an example of an equation which is valid in I_{DIF}, but not provable from E_{DIF}. The following version of *IF* is better from the viewpoint of ω-completeness:

module *IFa*
begin
include *IF*
variables $\sigma: \rightarrow$ **sorts**
$u, v, w: \rightarrow \sigma$
$X, Y, Z: \rightarrow bool$
equations $if(X, u, v) = if(X, u, if(\neg X, v, w))$ (3)
$if(X, u, if(Y, v, w)) = if(\neg X . Y, v, if(X, u, w))$ (4)
$if(X, u, if(Y, u, v)) = if(X \vee Y, u, v)$ (5)
$if(X, if(Y, u, v), w) = if(X . Y, u, if(X . \neg Y, v, w))$ (6)
$if(X, Y, Z) = (X . Y) + (\neg X . Z)$ (7)
end *IFa*.

THEOREM 3.3.1: $DIFa = D \cup IFa$ has the same initial algebra as *DIF* and is ω-complete.

PROOF: (*a*) $I_{DIFa} = I_{DIF}$, because $\Sigma_{DIFa} = \Sigma_{DIF}$ 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 (§3.2) because all *if*s can be eliminated from t by means of (7). 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

$$\delta_1$$

or

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

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) $\delta_i \neq \delta_j \quad (i \neq j)$
(ii) ξ_i is not of the form F or $\neg F$
(iii) $\xi_i . \xi_j =_{\mathbb{B}} F \quad (i \neq j)$
(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:

(S1) Eliminate all Boolean *if*s by means of (7).

(S2) Eliminate all *if*s from the second argument of other *if*s by means of (6).

(S3) Expand the innermost $if(\xi,\delta,\delta')$ (if it exists) into $if(\xi,\delta,if(\neg\xi,\delta',v)$ by means of (3). The resulting term satisfies (iv).

(S4) Merge all *if*s whose second argument contains the same constant or variable by means of (4) and (5). The resulting term satisfies (i).

(S5) If at this point the canonical form *in statu nascendi* is of the form

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

then turn it inside out, i.e. turn it by means of $\frac{n(n-1)}{2}$ applications of (4) into

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

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).

(S6) Bring all θ_i's into $\mathbb{B}$-canonical form ξ_i.

(S7a) If $\xi_i = F$ for some i, eliminate the corresponding *if* and δ_i by means of (1).

(S7b) If $\xi_i = \neg F$ for some i, the term is of the form $if(\neg F,\delta,v)$ because of property (iii) and (S7a). Reduce it to δ by means of (2). The resulting term satisfies (ii) and is in canonical form. □

Although, according to theorem 3.4.1, *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

$$S(if(X,x,y)) = if(X,S(x),S(y))$$
$$if(X,x,y).if(X,y,x) = x.y$$

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

module *IFb*
begin
  include *IFa*
  variables $X: \rightarrow bool$
    $\sigma,\tau: \rightarrow$ sorts
    $u,v: \rightarrow \sigma$
    $\Phi: \sigma \rightarrow \tau$
  equation $\Phi(if(X,u,v)) = if(X,\Phi(u),\Phi(v))$ (8)
end *IFb*.

Equation (8) is to be interpreted as follows. If *IFb* is combined with a specification S, (8) expands into n separate instances for every n-ary function $f \in \Sigma_{S \cup IFb}$ 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 (8) is $(f = if,\ k = 2)$

$$if(Y, if(X,u,v), w) = if(X, if(Y,u,w), if(Y,v,w)),$$

which is provable from E_{IFa}.

THEOREM 3.3.2: *IFb* is (weakly) ω-complete in the sense that $S \cup IFb$ 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 3.3.1, but with δ_i a term of sort s in S-canonical form. To bring a term into canonical form, follow steps (S1)-(S7b) of theorem 3.3.1 with two additional steps between (S1) and (S2), and a slightly different step (S4):

(S1.1) Move all *if*s to outermost positions by means of (8).

(S1.2) Bring all maximal *if*-free subterms (all of which are necessarily of the same sort) into S-canonical form.

(S4′) Merge all *if*s whose second argument contains syntactically identical S-canonical forms by means of (4) and (5). 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 3.3.2 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 $f, g: bool \rightarrow data$ and $h, k: bool \times bool \rightarrow data$. In that case some typical equations valid in $I_{S \cup IFb}$ but not provable from $E_{S \cup IFb}$ are

$$if(X, f(X), g(X)) = if(X, f(T), g(F)) \quad (9)$$
$$if(X+Y, h(X,Y), k(X,Y)) = if(X+Y, h(X,\neg X), k(X,X)) \quad (10)$$
$$if(X.Y, h(X,Y), k(X,Y)) = if(X.Y, h(T,T), if(X+Y, k(X,\neg X), k(F,F)). \quad (11)$$

Contrary to equations (1)-(8), which are valid in $I_{S \cup IF}$ for all S satisfying the sufficient completeness requirement just mentioned, equations like (9)-(11) are very much dependent on the particular S involved.

If interpreted as a left-to-right rewrite rule, equation (11) is typical of a whole class of rules whose right-hand sides contain more *if*s then 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 K, S: $\rightarrow F$
  $. : F \times F \rightarrow F$ (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 $x, y, z: \rightarrow F$
  equations $\mathrm{K}xy = x$
  $\mathrm{S}xyz = xz(yz)$
  S(S(KS)(S(KK)(S(KS)K)))(KK) = S(KK)
  S(KS)(S(KK)) = S(KK)(S(S(KS)(S(KK)(SKK)))(K(SKK)))
  S(K(S(KS)))(S(KS)(S(KS))) =
   = S(S(KS)(S(KK)(S(KS)(S(K(S(KS)))S))))(KS)
  S(S(KS)K)(K(SKK)) = SKK
end *CLX*.

CLX is identical to $\mathrm{CL}+\mathrm{A}_{\beta\eta}$ in Barendregt [16]. Hence, according to [16], 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

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

then also

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

Is *CLX* ω-*extensional*? That is, does

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

imply

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

Plotkin has shown that the $\lambda K\beta\eta$-calculus is not ω-extensional ([16], theorem 17.3.30). Hence, *CLX* is not ω-extensional either. Because

$$\omega\text{-completeness} \wedge \text{extensionality} \Rightarrow \omega\text{-extensionality}, \qquad (1)$$

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 (1) plus the fact that *CLX* is combinatorially complete, it is enough to show that

$$\text{combinatorial completeness} \wedge \omega\text{-extensionality} \Rightarrow \omega\text{-completeness}. \qquad (2)$$

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

$$E_{CLX} \vdash f = \phi x_1 \cdots x_k, \; g = \psi x_1 \cdots x_k$$

Applying ω-extensionality k times gives

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

Hence

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

and

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

This proves (2).

Two questions I have not succeeded in answering are:

OPEN QUESTION 3.4.1: Does *CLX* have an ω-complete version (with or without auxiliary sorts and functions)?

OPEN QUESTION 3.4.2: Are the open equations valid in the initial algebra of *CLX* recursively enumerable?

If - as would be my guess - the answer to the second question is *no*, the answer to the first question must also be *no* according to theorem 2.4.

ACKNOWLEDGEMENTS

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

REFERENCES

1. Futamura, Y., "Partial evaluation of computation process - an approach to a compiler-compiler", *Systems-Computers-Controls*, **2**(1971), 5, pp. 45-50.
2. Beckman, L., Haraldson, A., Oskarsson, O., & Sandewall, E., "A partial evaluator, and its use as a programming tool", *Artificial Intelligence*, **7**(1976), pp. 319-357.
3. Ershov, A.P., "Mixed computation: potential applications and problems for study", *Theoretical Computer Science*, **18**(1982), pp. 41-67.
4. Komorowski, H.J., *A Specification of an Abstract PROLOG Machine and its Application to Partial Evaluation*, Dissertation No. 69, Linköping University, 1981.
5. Komorowski, H.J., "Partial evaluation as a means for inferencing data structures in an applicative language: a theory and implementation in the case of PROLOG", *Conference Record of the 9th Annual ACM Symposium on Principles of Programming Languages*, ACM, 1982, pp. 255-267.
6. Huet, G., & Oppen, D.C., "Equations and rewrite rules: a survey", in: Book, R., (ed.), *Formal Languages: Perspectives and Open Problems*, Academic Press, 1980.
7. Goguen, J.A. & Meseguer, J., "An initiality primer", in: Nivat, M., & Reynolds, J., (eds), *Application of Algebra to Language Definition and Compilation*, North-Holland, 1983.
8. Huet, G., & Hullot, J.M., "Proofs by induction in equational theories with constructors", *Journal of Computer and System Sciences*, **25**(1982), pp. 239-266.
9. Lyndon, R.C., "Identities in finite algebras", *Proceedings of the American Mathematical Society*, **5**(1954), pp. 8-9.
10. Grätzer, G., *Universal Algebra*, 2nd ed., Springer-Verlag, 1979.
11. Bergstra, J.A., & Tucker, J.V., "Algebraic specifications of computable and semi-computable data structures", Report IW 115/79, Department of Computer Science, Centre for Mathematics and Computer Science, Amsterdam, 1979.
12. Bergstra, J.A., & Tucker, J.V., "Initial and final algebra semantics for data type specifications: two characterization theorems", *SIAM Journal on Computing*, **12**(1983), 2, pp. 366-387.

13. Huet, G., "Confluent reductions: abstract properties and applications to term rewriting systems", *Journal of the ACM*, **27**(1980), pp. 797-821.
14. Davis, M., Matijasevic, Y. & Robinson, J., "Hilbert's tenth problem: positive aspects of a negative solution", in: Browder, F.E., (ed.), *Mathematical Developments Arising from Hilbert Problems*, American Mathematical Society, 1976, pp. 323-378.
15. Hsiang, J., *Topics in Automated Theorem Proving and Program Generation*, Report UIUCDCS-R-82-1113, Department of Computer Science, University of Illinois at Urbana-Champaign, 1982.
16. Barendregt, H.P., *The Lambda Calculus*, North-Holland, 1981.