COMPLETENESS RESULTS FOR THE EQUIVALENCE OF RECURSIVE SCHEMAS

by

Bruno Courcelle and Jean Vuillemin

Memorandum No. ERL-M485

December 1974

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

COMPLETENESS RESULTS FOR THE EQUIVALENCE OF RECURSIVE SCHEMAS

by

Bruno Courcelle and Jean Vuillemin

Abstract

We investigate the possibilities of automating equivalence proofs for recursive schemas. A formal proof system, adapted from deBakker-Scott [4], is shown to be complete with respect to provability of equivalence of pure recursive schemes. The result is obtained by showing the correspondence between operational and denotational semantics of a simple recursive language.

Keywords: Proofs of programs properties. Automatic program verification. Operational and denotational semantics. Fixed points. Deterministic simple languages.

Research sponsored by the National Science Foundation Grant GJ-43318.

1. Introduction

The theory of program schemas, as illustrated for example in [1], [11], [14], [16] or [22] has been mainly motivated by the study of sequential Algol or Fortran like programs. Within the framework of more general semantic theories, such as those of Scott [23], where one also considers parallel programs or infinite loops with interesting side-effects (as, for instance, a program enumerating the prime numbers), the results about program schemas have a different flavor, as shown for example in [2], [7], [21] or [25].

In this work, we are more interested in results regarding tools for formal description of programming languages, than in the more traditional decidability questions for equivalence of program schemes.

A programming language and its semantics are described, in the manner of Scott and Strachey [28]. By semantics, we mean here what has been called denotational semantics, i.e., a mapping between programs and their intended mathematical meaning as opposed to operational semantics, i.e., some kind of interpreter for the language. In the usual practice of Mathematical Logic, the first one would be called semantics and the other one syntax.

In [8] a canonical domain of interpretation, akin to the Herbrand universe for first order theories is constructed, and it is proved that (Theorem 1) semantic properties are true if and only if they are true in this particular interpretation. Using this result as a basic tool, we exhibit a normal representation for programs, and show that equivalence between recursive program schemes is decidable. It is also possible to obtain this decidability result via a coding of schemas into deterministic simple languages, where equivalence was shown to be decidable by

Hopcroft-Korenjak [13].

Using this normal representation, we show (Theorem 2) that the semantic definition used is the best possible compatible with evaluation rules such as the copy rule. More precisely, we show that, in the terminology of Milner [19], the semantics is both adequate and fully-abstract with respect to the copy rule. Another way to put this is that the semantics chosen is compatible with all operational semantics for the language, and it is not more than this because all terms which are semantically equivalent are computationally equivalent with respect to a particular representation of the program.

Finally, we introduce a formal proof system, adapted from [4] and [18], and prove (Theorem 3) that all equivalences between monadic terms or programs which are semantically valid are (effectively) provable within the formal system. With minor modifications, this result is shown in [9] to also apply to the formal proof systems studied by deBakker [2], who has already obtained a similar result for a smaller class of languages: we show (in [9]) that a slight extension of deBakker's proof system is complete with respect to provability of equivalence within the class of free recursive program schemes studied by deBakker-Scott [4] and Ashcroft-Manna-Pnueli [1].

2. Syntax and Semantics of the Language

2.1 Syntax

In an approximation of BNF notation, the syntax of our programming language is defined by:

$$\text{<program>} ::= \begin{cases} F_1(X) \Leftarrow \text{<term 1>} \\ \vdots \\ F_k(X) \Leftarrow \text{<term k>} \end{cases}$$

$$\begin{aligned} \text{<term>} ::= X &| g_1(\text{<term 1>},\ldots,\text{<term } p_1\text{>}) \\ &| g_2(\text{<term 1>},\ldots,\text{<term } p_2\text{>}) \\ &\vdots \\ &| g_h(\text{<term 1>},\ldots,\text{<term } p_h\text{>}) \\ &| F_1(\text{<term>}) \\ &\vdots \\ &| F_k(\text{<term>}) \end{aligned}$$

Typical examples are:

$$F_1(X) \Leftarrow g_1(X, F_1F_1(X))$$

or

$$\begin{cases} F_1(X) \Leftarrow g_1(X, F_2(X)) \\ F_2(X) \Leftarrow g_2(F_2(X), F_1(X)) \end{cases},$$

where standard conventions regarding omission of parenthesis are applied. Among the program thus defined are "parameterless" programs, such as $F(X) \Leftarrow F(X)$ or $F(X) \Leftarrow g_1(F(X),F(X))$, which we want to exclude (at "compile time") from our language. Let us therefore associate to each term T in the context of a program P: $\langle F_1(X) \Leftarrow P_1,\ldots,F_k(X) \Leftarrow P_k\rangle$ a natural number $\delta(T)$ as follows:

(i) $\delta(X) = 0$.

(ii) $\delta(g_i(T_1,\ldots,T_{p_i})) = 1 + \min\limits_{1\le j\le p_i} (\delta(T_j))$.

(iii) $\delta(F_j(T)) = \delta(P_j\{T\})$.

Here $P\{T\}$ denotes the result of substituting T for all occurrences of X in P. Whenever $\delta(T)$ is not defined, we say that $\delta(T) = \infty$. For example:

if $F(X) \Leftarrow g_1(X,Fg_2(X))$ then $\delta(F(X)) = 1$ and $\delta(FF(X)) = 2$ while,

if $F(X) \Leftarrow g_1(F(X),F(X))$ then $\delta(F(X)) = \infty$ and $\delta(g_1(F(X),X)) = 1$.

This use of the symbol ∞ is legitimate since it is clearly decidable whether $\delta(T) = \infty$ or not for any term T and program P.

From now on, we shall only concern ourselves with programs which satisfy the following syntactic restriction:

Definition: A program $P = \langle F_1(X) \Leftarrow P_1, \ldots, F_k(X) \Leftarrow P_k \rangle$ is acceptable if and only if $\delta(F_i(X)) \neq \infty$ for all $1 \leq i \leq k$.

It will also be convenient to insist that $\delta(F_i(X)) \neq 0$ for all i in acceptable programs.

It is easily seen that, if P is acceptable, then any subterm T' of an arbitrary term T has a finite $\delta(T')$, and

Lemma 1: For any terms T and T', $\delta(T\{T'\}) = \delta(T) + \delta(T')$.

Proof: The proof is by induction on $\delta(A)$ where $A = T\{T'\}$.

If $T = X$ then $\delta(T) = 0$, $A = T'$ and $\delta(A) = \delta(T') = \delta(T) + \delta(T')$.

If $T = g(T_1,\ldots,T_p)$ then $A = g(A_1,\ldots,A_p)$ where $A_j = T_j\{T'\}$ for $1 \leq j \leq p$. By definition of δ, we know that there exists j_o and j_1 in $[1,p]$ such that:

(i) $\delta(T) = \delta(T_{j_o}) + 1 \leq \delta(T_{j_1}) + 1$ and

(ii) $\delta(A) = \delta(A_{j_1}) + 1 \leq \delta(A_{j_o}) + 1$.

Since $\delta(A_{j_1}) < \delta(A)$ and $\delta(T_{j_o}) + \delta(T') < \delta(T) + \delta(T') = \delta(A)$, we know by induction that $\delta(A_{j_o}) = \delta(T_{j_o}) + \delta(T')$ and $\delta(A_{j_1}) = \delta(T_{j_1}) + \delta(T')$. It then follows from (i) and (ii) that $\delta(A) = \delta(T) + \delta(T')$.

If $T = F(T_1)$, then it is easily seen that $\delta F \neq \infty$ implies that

$\delta(T) = \delta(T')$ for some $T' = g(T'_1,\ldots,T'_p)$ and we are back to the previous case. □

2.2 Semantics

Let us first recall some definitions:

Definition: A complete partial order (c.p.o.) is a set D together with a partial order $\subseteq$ over D such that

(i) D has a minimal element $\perp_D$.

(ii) Every chain $x_1 \subseteq x_2 \subseteq \cdots \subseteq x_n \subseteq \cdots$ has a least upper (l.u.b.) $\bigcup_{i\geq 0} x_i$.

Actually, Scott [26] uses complete lattices instead of c.p.o.'s but c.p.o.'s seem to be easier to work with, and have been used for example in [8], [10], [18], [21], [23] and [31]. Discussions regarding the choice of c.p.o.'s versus complete latrices are given in [10], [23] and [31].

Definition: A mapping $f: D \to D'$ between two c.p.o.'s D and D' is continuous if, for every chain $X \subseteq D$, we have $\cup f(X) = f(\cup X)$.

Note that every continuous function is also monotone, i.e. $x \subseteq y$ implies $f(x) \subseteq f(y)$.

We denote by $[D \to D']$ the set of continuous functions between the c.p.o.'s D and D'. For polyadic functions, continuity it meant component-wise.

Lemma 2: If D and D' are c.p.o.'s then $[D \to D']$ is a c.p.o.

Lemma 3: Every continuous $f \in [D \to D]$ has a least fixed point $Y(f) \in D$ and $Y(f) = \bigcup_{n \geq 0} f^n(\perp)$.

We can now define the semantic interpretation of our programming language. We shall only do so for programs with only one recursively defined function F. The generalization to mutually recursive systems is straightforward.

Definition: An interpretation I of a program P consists of

(i) A complete partial order D.

(ii) To each function symbol g_i with arity $p_i \geq 0$ is associated a function $g_i \in [D^{p_i} \to D]$.

Given an interpretation I, we construct the semantics $\mathcal{S}[\![T]\!]$ of the term T in the context of the program P as follows: given a meaning $f \in [D \to D]$ and $a \in D$ for the variable names F and X respectively, we define the interpretation $\mathcal{S}[\![T]\!]\{f/F, a/X\}$ inductively as follows:

(i) $\mathcal{S}[\![X]\!]\{f/F, a/X\} = a$.

(ii) $\mathcal{S}[\![g(T_1,\ldots,T_p)]\!]\{f/F, a/X\} = g(t_1,\ldots,t_p)$ where g is the (continuous) function associated to g in the interpretation and $t_i = \mathcal{S}[\![T_i]\!]\{f/F, a/X\}$ for $1 \leq i \leq p$.

(iii) $\mathcal{S}[\![F(T)]\!]\{f/F, a/X\} = f(\mathcal{S}[\![T]\!]\{f/F, a/X\})$.

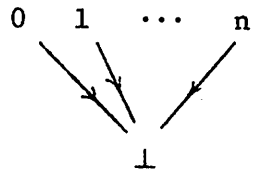
For the sake of readability, we abbreviate $\lambda a.\mathcal{S}[\![T]\!]\{f/F, a/X\}$ by $\mathcal{S}[\![T]\!]\{f/F\}$ and $\lambda f.\mathcal{S}[\![T]\!]\{f/F\}$ by $\mathcal{S}[\![T]\!]$.

Lemma 4: For any term T, $\mathcal{S}[\![T]\!] \in [[D \to D] \to [D \to D]]$ and we denote by $Y(\mathcal{S}[\![T]\!])$ the least fixed point of $\mathcal{S}[\![T]\!]$.

In the context of a program $F(X) \Leftarrow P$, the interpretation $I[[T]]$ of a term T is then defined as

$$I[[T]] = \mathcal{S}[[T]]\{f_P/F\} \quad \text{where} \quad f_P = Y(\mathcal{S}[[P]]) \quad .$$

For example, if the domain of interpretation is



and $F(X) \Leftarrow g_1(X, g_2(X, Fg_3(X)))$, with $g_1(x,y) =$ *if* $x = 0$ *then* 1 *else* y and $g_2(x,y) = x*y$ (x times y) then $I[[F(X)]] = \lambda x.x!$ (x factorial). If the domain of interpretation is the set of real intervals $[x,x']$ with $0 \leq x \leq x' \leq \infty$ and $x, x' \in \mathbb{R}$, ordered by $[x,x'] \subseteq [y,y']$ if and only if $0 \leq x \leq y \leq y' \leq x' \leq \infty$ (see [26] and [31]), and $F(X) \Leftarrow h(F(X),X)$ with $h(x,y) = 1 + \frac{y-1}{x+1}$, then $I[[F(X)]] = \lambda x.\sqrt{x}$ is the square root function, for $x \geq 1$.

If P is a program and I an interpretation, we write $P \models_I T \equiv T'$ for $I[[T]] = I[[T']]$.

Definition: Let P be a program, T and T' two terms. We say that $P \models T \equiv T'$ if and only if $P \models_I T \equiv T'$ for all interpretations I. We say that two programs $P = \langle F_1(X) \Leftarrow P_1, \ldots, F_k(X) \Leftarrow P_k \rangle$ and $P' = \langle F'_k(X) \Leftarrow P'_1, \ldots, F'_{k'}(X) \Leftarrow P_{k'} \rangle$ are equivalent if and only if $P, P' \models F_1(X) \equiv F'_1(X)$.

In [8], a canonical interpretation H is constructed which has the property:

Theorem 1: For any terms T, T' and program P, the equality $P \models T \equiv T'$ holds if and only if $P \models_H T \equiv T'$.

Proof: (See [8]). It is convenient to view elements of H as infinite formulae. For example, if $F(X) \Leftarrow g(X,F(X))$, then $H[\![F(X)]\!]$ "behaves" like $g(X,g(X,g(X,\ldots)\ldots)$ ad-infinitum. □

For example, if $P = \langle F_1(X) \Leftarrow g(X,F_1(X)),\ F_2(X) \Leftarrow g(X,g(X,F_2(X)))\rangle$, then $P \models F_1(X) \equiv F_2(X)$ and $P \models F_2(X) \equiv g(X,F_1(X))$ are valid, while $P \models g(X,X) \equiv k(X)$ is not valid. More generally, it is easily shown that:

Corollary: $P \models g_i(T_1,\ldots,T_{p_i}) \equiv g_j(T'_1,\ldots,T'_{p_j})$ *if and only if* $i = j$ *and* $P \models T_k \equiv T'_k$ *for* $1 \leq k \leq p_i$.

It is also proved in [8] or [31] that the following properties of left and right simplifications holds:

Lemma 5: *For any terms* T, T', T'' *and acceptable program* P, $P \models T\{T''\} \equiv T'\{T''\}$ *if and only if* $P \models T''\{T\} \equiv T''\{T'\}$ *if and only if* $P \models T \equiv T'$.

Note that this is not true if P is not acceptable, as shown by the counterexample: $P = F(X) \Leftarrow G(F(X))$, $T = X$, $T' = T'' = F(X)$.

3. Decidability of Equivalence Between Acceptable Programs

Let $P = \langle F_1(X) \Leftarrow P_1,\ldots,F_k(X) \Leftarrow P_k\rangle$ be an acceptable program. We associate to P an equivalent program P' as follows: first, each P_i of the form $F_j(T)$ is replaced by $P_j\{T\}$ until all P_i's are of the form $g_j(T_1,\ldots,T_{p_j})$. (This is possible because P is acceptable). Then, to each proper-subterm of the P_i's of the form $g_n(T_1,\ldots,T_{p_n})$, we associate a different name, $F_{k+1}(X)$ through $F_m(X)$, replace in P_i the subterm by the corresponding $F_j(X)$, and add the defining equation $F_j(X) \Leftarrow g_n(T_1,\ldots,T_{p_n})$. We

obtain a new program $P' = \langle F_1(X) \Leftarrow P'_1, \ldots, F_m(X) \Leftarrow P'_m \rangle$ where each P'_i is of the form $G_j(M_1, \ldots, M_{p_j})$ with $M_q = F_{i_1} \cdots F_{i_s}(X)$. We say that such a program P' is an standard form, and it is easily seen that P and P' are equivalent.

For example,

$$\text{if } P = \begin{cases} F_1(X) \Leftarrow F_2F_2(X) \\ F_2(X) \Leftarrow g(X, F_1(X)) \end{cases} \text{ then } P' = \begin{cases} F_1(X) \Leftarrow g(F_2(X), F_1F_2(X)) \\ F_2(X) \Leftarrow g(X, F_1(X)) \end{cases}$$

and, if $P = \langle F_1(X) \Leftarrow g(g(X, F_1(X)), F_1g(X, F_1(X))) \rangle$ then

$$P' = \begin{cases} F_1(X) \Leftarrow g(F_2(X), F_1F_3(X)) \\ F_2(X) \Leftarrow g(X, F_1(X)) \\ F_3(X) \Leftarrow g(X, F_1(X)) \end{cases}$$

To be precise, we should use different symbol letters, say F'_1 through F'_m for the variables of P' but no confusion can arise here since $P, P' \models F_i(X) \equiv F'_i(X)$ for $1 \leq i \leq k$.

We can now concentrate our attention to programs in standard form. As a notational convenience, we reserve the letter M for terms of the form $F_{i_1} \cdots F_{i_s}(X)$.

Before describing an algorithm for deciding the equivalence between standard programs, thus between acceptable programs, let us establish two preliminary results:

Lemma 6: The set of valid formulae of the form $P \models F_i(X) \equiv M$ is finite and the letter F_i appears in M if and only if $M = F_i(X)$.

Proof: By Lemma 1, if $M = F_{i_1} \cdots F_{i_s}(X)$ then $\delta M = \sum_{1 \leq j \leq s} \delta(F_{i_j}(X))$. Since $\delta(F_{i_j}(X)) \geq 1$, the number of valid equalities $P \models F_i(X) \equiv M$

is bounded by, say, $\binom{\delta(F_i(X))}{k-1}$, where k is the number of F_i's in P. Furthermore, if F_i occurs in M, it must be alone. □

Equations of the form $P \models F_i(X) \equiv M$ will be called <u>elementary equations</u>.

<u>Lemma 7</u>: <u>Let</u> P <u>be in standard form. If any equality</u> $P \models F_i(M_1) \equiv F_j(M_2)$ <u>holds, with</u> $\delta(F_i) \leq \delta(F_j)$, <u>we can find</u> M_3 <u>such that</u>:

$$P \models F_j(X) \equiv F_i(M_3) \text{ and } P \models M_1 \equiv M_3\{M_2\} \quad .$$

<u>Proof</u>: We prove the more general result $[P \models M_i\{M_1\} \equiv M_j\{M_2\}$ and $\delta(M_i) \leq \delta(M_j]$ implies $[P \models M_j \equiv M_i\{M_3\}$ and $P \models M_1 \equiv M_3\{M_2\}]$ by induction on $\delta(M_i)$.

If $\delta(M_i) = 0$, i.e., $M_i = X$ we choose $M_3 = M_j$.

Otherwise, $M_i = F_i(M_i')$ and, since $\delta(M_j) \geq \delta(M_i)$, we must also have $M_j = F_j(M_j')$. Let $P_i = g(M_{i_1},\ldots,M_{i_p})$ and $P_j = g(M_{j_1},\ldots,M_{j_p})$ be the defining right-hand-side of $F_i(X)$ and $F_j(X)$ respectively in P. Note that the two g's are identical because of the corollary of Theorem 1. For the same reason, $P \models M_i\{M_1\} \equiv M_j\{M_2\}$ implies $P \models M_{i_k}\{M_i'\{M_1\}\}$ $\equiv M_{j_k}\{M_j'\{M_2\}\}$ for all $1 \leq k \leq p$. Let us choose k so that $\delta(M_{i_k}) = \delta(P_i) - 1$. It follows from Lemma 1 that $\delta(M_{i_k}\{M_i'\}) = \delta(P_i)$ $+ \delta(M_i') - 1 = \delta(M_i) - 1$.

By induction, we can therefore find M_3 such that $P \models M_1 \equiv M_3\{M_2\}$. It follows that $P \models M_i\{M_1\} \equiv M_i\{M_3\{M_2\}\} \equiv M_j\{M_2\}$ and, by Lemma 5, $P \models M_i\{M_3\} \equiv M_j$. □

The reader who is familiar with the results of Hopcroft and Korenjak [13] about simple deterministic languages has noticed the similarity

between this last result and some of the Lemmas in [13]. The correspondence between our schemas and deterministic simple languages is explicited in [6] and results obtained in each formalism can be translated into the other.

We can now describe an algorithm for deciding if $P \models M \equiv M'$, where P is any program in standard form. The structure of the algorithm is:

start: simplify; generate new equations; go to start;

In the simplification part, all equations $M \equiv M'$ which are not elementary (i.e., of the form $F_i(X) \equiv M$) are replaced by a set of elementary equations through repeated use of Lemma 7. Once all equations are elementary, we replace $F_i(X) \equiv F_j(M')$ by $G_i(M_{i_1},\ldots,M_{i_{p_i}}) \equiv G_j(M_{j_1}\{M'\},\ldots,M_{j_{p_j}}\{M'\})$. We assume here that $P = \langle F_m(X) \Leftarrow G_m(M_{m_1},\ldots,M_{m_{p_m}})$ for $1 \leq m \leq k\rangle$.

If $i \neq j$, we terminate with a no answer. Otherwise, we replace this equation by $M_{i_1} \equiv M_{j_1}\{M'\},\ldots,M_{i_{p_i}} \equiv M_{j_{p_j}}\{M'\}$.

The algorithm terminates with a yes answer when no new elementary equation is found after the simplification stage (the number of such equations is finite by Lemma 6); it terminates with a no answer when a false equation $P \models g_i(M_1,\ldots,M_{p_i}) \equiv g_j(M'_1,\ldots,M'_{p_j})$ with $i \neq j$ is generated at either stage.

For example, if

$$P = \begin{cases} F_1(X) \Leftarrow g(X,F_3(X)) \\ F_2(X) \Leftarrow g(F_1(X),F_2F_2(X)) \\ F_3(X) \Leftarrow g(F_2(X),F_3F_2(X)) \end{cases}$$

The algorithm, with input $F_2 \equiv F_1F_1$ proceeds as follows:

$$F_2 \equiv F_1F_1 \xrightarrow{\text{simplify}} F_2 \equiv F_1F_1 \xrightarrow{\text{generate}} F_1 \equiv F_1,$$

$$F_2F_2 \equiv F_3F_1 \xrightarrow{\text{simplify}} F_1 \equiv F_1,\ F_2 \equiv F_1F_1,$$

$$F_3 \equiv F_2F_1 \xrightarrow{\text{generate}} F_3 \equiv F_3,\ F_1 \equiv F_1,\ F_2F_2 \equiv F_3F_1,\ F_2 \equiv F_1F_1,$$

$$F_3F_2 \equiv F_2F_2F_1 \xrightarrow{\text{simplify}} F_3 \equiv F_3,\ F_1 \equiv F_1,\ F_2 \equiv F_1F_1,\ F_3 \equiv F_2F_1 .$$

The algorithm terminates here with a yes answer since no new elementary equation (except the trivial one $F_3 \equiv F_3$) is generated. Of course, this is a crude presentation and there are many ways in which we could improve the efficiency of this algorithm.

4. Equivalence Between Operational and Denotational Semantics

We now define a canonical representation for programs and for terms. Let us consider a standard program $P = \langle F_1(X) \Leftarrow P_1, \ldots, F_m(X) \Leftarrow P_m \rangle$ and valid equation $P \models F_i(X) \equiv M$ which is not of the uninteresting form $P \models F_i \equiv F_i$. Any such equation can be used to "simplify" P as follows: for all $j \neq i$, replace $F_j(X) \Leftarrow P_j$ by $F_j(X) \Leftarrow P_j\{M/F_i\}$ where $P_j\{M/F_i\}$ denotes the result of substituting in P_j all subterms of the form $F_k(M_k)$ by $M\{M_k\}$. Replace then $F_i(X) \Leftarrow P_i$ by $F_i(X) \Leftarrow M$. We say that a variable F_i which does appear in any of the right-hand-sides of the defining equations of a program P is active. In this sense, the variable F_i is no longer active in the simplified program P. We can perform simplifications in such a way that the defining equations $F_j(X) \Leftarrow P_j$ are of two forms: either F_j is active and $P_j = G_j(M_1, \ldots, M_{P_j})$ or F_j is not active and $P_j = M$, in both cases, M and the M_i's are of the form $F_{i_1} \cdots F_{i_s}(X)$ where the F_{i_k} are active variables.

Definition: A canonical representation $\bar{P}$ of P is, among all such representations, one for which the number of active variables is minimal and such that the sum of the indices of active variables is also minimal.

For example, if

$$P = \begin{cases} F_1(X) \Leftarrow g(X, F_3(X)) \\ F_2(X) \Leftarrow g(F_1(X), F_2F_2(X)) \\ F_3(X) \Leftarrow g(F_2(X), F_3F_2(X)) \end{cases}$$

then

$$\bar{P} = \begin{cases} F_1(X) \Leftarrow g(X, F_1F_1F_1(X)) \\ F_2(X) \Leftarrow F_1F_1(X) \\ F_3(X) \Leftarrow F_1F_1F_1(X) \end{cases} ;$$

if

$$P = \begin{cases} F_1(X) \Leftarrow g(F_2(X), F_1F_3(X)) \\ F_2(X) \Leftarrow g(X, F_1(X)) \\ F_3(X) \Leftarrow g(X, F_1(X)) \end{cases}$$

then

$$\bar{P} = \begin{cases} F_1(X) \Leftarrow F_2F_2(X) \\ F_2(X) \Leftarrow g(X, F_2F_2(X)) \\ F_3(X) \Leftarrow F_2(X) \end{cases} .$$

Of course, we must prove that $\bar{P}$ does not depend upon the order in which eliminations are performed, i.e.:

Lemma 8: The canonical representation $\bar{P}$ of a program P is unique.

Proof: Assuming the existence of two canonical representations $\bar{P}_1$ and $\bar{P}_2$ of some program P, we prove that they are identical.

First we notice that the active variables must be the same in $\bar{P}_1$ and $\bar{P}_2$; otherwise, let us consider an i such that F_i is active in $\bar{P}_1$ and not $\bar{P}_2$. Therefore $\bar{P}_2 \models F_i \equiv F_{j_1} \cdots F_{j_s}$ hence $\bar{P}_1 \models F_i \equiv F_{j_1} \cdots F_{j_s}$. If any of the F_{j_k} is not active in $\bar{P}_1$, we can replace it by some $M = F_{n_1} \cdots F_{n_p}$ where the F_{n_k} are active in P_1. It follows that $\bar{P}_1 \models F_i \equiv M'$ where $M' = F_{k_1} \cdots F_{k_r}$ with the F_{k_j} active in $\bar{P}_1$. But this is impossible, unless $M' = F_i$, otherwise, we could simplify $\bar{P}_1$ further and reduce the number of active equations by one. If $M' = F_i$, this implies that the defining equation of F_i in $\bar{P}_2$ was $F_i(X) \Leftarrow F_j(X)$, with F_j active in $\bar{P}_2$. If $i < j$ we can use F_i instead of F_j as active variable in $\bar{P}_2$; but this is impossible since this would decrease the sum of the indices of active variables. If $j > i$, then $\bar{P}_1 \models F_i \equiv F_j$ and we could use F_j instead of F_i as active variable in $\bar{P}_1$; this is again a contradiction since it would decrease the sum of the indices of active variables in $\bar{P}_1$. It follows that active variables are the same in $\bar{P}_1$ and $\bar{P}_2$. We still have to show that the defining terms are the same in $\bar{P}_1$ and $\bar{P}_2$. For this purpose, it is sufficient to prove that, in a canonical representation $\bar{P}$ of P, valid equations $\bar{P} \models M \equiv M'$ where M and M' are of the form $F_{i_1} \cdots F_{i_s}(X)$ with F_{i_k} active are of the trivial form $M \equiv M$, that is $\bar{P} \models M \equiv M'$ if and only if $M = M'$.

This is easy by induction on $\delta(M)$: if $\delta(M) = 0$, then $\delta(M') = 0$ and $M = M' = X$. Otherwise, $M = F_i(M_1)$ and $M' = F_j(M'_1)$ with say $\delta F_i \leq \delta F_j$. If i where different from j, we could use Lemma 7 and generate an equation $F_j(X) \equiv F_i(M_3)$ which could be used to reduce by one the number of active variables in $\bar{P}$. Therefore $i = j$, and by Lemma 5, $\bar{P} \models M_1 \equiv M'_1$. By induction, this implies $M_1 = M'_1$ hence $M = M'$. □

This representation is only canonical with respect to a given program, and not to all equivalent programs. For example,

$$\bar{P}_1 = \begin{cases} F_1(X) \Leftarrow g_1(X, F_2F_3(X)) \\ F_2(X) \Leftarrow g_2(F_2(X), X) \\ F_3(X) \Leftarrow g_3(X, F_3(X)) \end{cases} \quad \text{and} \quad \bar{P}_2 = \begin{cases} F_1(X) \Leftarrow g_1(X, F_2(X)) \\ F_2(X) \Leftarrow g_2(F_2(X), F_3(X)) \\ F_3(X) \Leftarrow g_3(X, F_3(X)) \end{cases},$$

which are both in canonical form are equivalent yet not identical.

Using the results of Courcelle [5], [6] about simple languages and recursive schemes, it is easy to construct such a canonical representation, but this will not be necessary for the purpose of this paper.

We now define a canonical representation for terms (in the context of a given program P):

Definition: A term T is said to be reduced with respect to the program P if it has no subterm $T' = G_i(T_1, \ldots, T_{p_i})$ such that $P \models T' \equiv F_j(R)$ for some term R and variable F_j.

Lemma 9: To each term T and program P, we can associate a unique term $\bar{T}$ reduced with respect to P, which we call the canonical representation $\bar{T}$ of T.

Proof: We prove that $\bar{T}$ exists and that it is unique.

First, we eliminate in T all variables F_i which are not active in $\bar{P}$; this is done by replacing T by $T\{M/F_i\}$ where F_i is defined by $F_i(X) \Leftarrow M$ in $\bar{P}$. A term T' such that $\bar{P} \models T' \equiv T$ is thus obtained, in which all variables which do occur are active. Let us show that there exist $\bar{T}$ which is reduced with respect to $\bar{P}$ and such that $\bar{P} \models T' \equiv \bar{T}$ hence $\bar{P} \models \bar{T} \equiv T$.

The proof is by induction on the structure of T. If $T = X$ then $\bar{T} = X$ is reduced. If $T = F_i(T')$ then we know by induction that $\bar{T}'$ exists and we take $\bar{T} = F_i(\bar{T}')$. If $T = g_i(T_1,\ldots,T_{p_i})$, we know by induction that $\bar{T}_1,\ldots,\bar{T}_{p_i}$ exist. Let $T' = g_i(\bar{T}_1,\ldots,\bar{T}_{p_j})$. Suppose that $\bar{P} \models T' \equiv F_j(R)$ for some R and active F_j. The defining equation $F_j(X) \Leftarrow P_j$ of F_j in $\bar{P}$ must be of the form $P_j = g_i(M_1,\ldots,M_{p_i})$, and therefore $\bar{P} \models \bar{T}_i \equiv M_i\{R\}$ for $1 \leq i \leq p_i$. It is easy to see (using Lemma 7 again) that such an equation implies that $\bar{T}_i = M_i\{\bar{T}'_i\}$ with $\bar{P} \models \bar{T}'_i \equiv R$ and $\bar{T}'_i$ reduced with respect to $\bar{P}$. We can then take $\bar{T} = F_j(T'_i)$.

In order to prove that $\bar{T}$ is unique let us consider $\bar{T}_1$ and $\bar{T}_2$ and prove by induction on the structure of $\bar{T}_1$ that $\bar{P} \models \bar{T}_1 \equiv \bar{T}_2$ if and only if $\bar{T}_1 = \bar{T}_2$. If $\bar{T}_1 = X$ then $\bar{T}_2$ must also be equal to X. If $\bar{T}_1 = F_i(\bar{T}'_1)$ then $\bar{T}_2$ cannot be of the form X or $G(T''_1,\ldots,T''_p)$, otherwise it would not be reduced. It is therefore of the form $\bar{T}_2 = F_i(\bar{T}'_2)$ and $j = i$ otherwise, by Lemma 7, $\bar{P}$ would not be canonical. By Lemma 5, $\bar{P} \models \bar{T}'_1 \equiv \bar{T}'_2$ and, because of the induction hypothesis; $\bar{T}'_1 = \bar{T}'_2$ hence $\bar{T}_1 = \bar{T}_2$. If $\bar{T}_1 = G(\bar{T}_{1,1},\ldots,\bar{T}_{1,p})$, then $\bar{T}_2 = G(\bar{T}_{2,1},\ldots,\bar{T}_{2,p})$ otherwise $\bar{T}_1$ would not be reduced and the argument carries through by induction. □

In order to relate operational and denotational semantics, we need to define the computation rules in our language. For this purpose, we say that $T_1 \xrightarrow{P} T_2$ whenever T_2 is the result of replacing some subterms $F_j(T')$ of T_1 by $P_j\{T'\}$ in T_1, where P_j is the right-hand-side corresponding to the definition of F_j in P. Let $\xrightarrow{*}_P$ be the reflexive and transitive closure of $\xrightarrow{P}$. This relation, which models Algol's copy rule, has been studied by Vuillemin [30] who showed that the set $\{T' \mid T \xrightarrow{*}_P T'\}$

is a lattice, called the computation lattice of T according to P.

The relation $\overset{*}{\to}$ induces an equivalence relation (see [31]) over the set of terms:

Definition: We say that two terms T_1 and T_2 are interconvertible: $T_1 \overset{*}{\underset{P}{\leftrightarrow}} T_2$ with respect to P if there exists T_3 such that $T_1 \overset{*}{\underset{P}{\to}} T_3$ and $T_2 \overset{*}{\underset{P}{\to}} T_3$.

This syntactical notion of equivalence between terms can be used to characterize semantic equivalence in the following sense:

Theorem 2: For any acceptable program P and term T, the set $\{T' \mid P \models T \equiv T'\}$ of terms semantically equivalent to T coincide with the set $\{T' \mid T \overset{*}{\underset{\bar{P}}{\leftrightarrow}} T'\}$ of terms interconvertible to T with respect to the canonical representation $\bar{P}$ of P.

Proof: First, we notice that $T \overset{*}{\underset{\bar{P}}{\to}} T_a$ where T_a is obtained by replacing in T the variables which are inactive in $\bar{P}$. We then prove that the set $\{T' \mid P \models T_a \equiv T'\}$ of terms equivalent to T_a coincide with the computation lattice $\{T' \mid \bar{T}_a \overset{*}{\underset{\bar{P}}{\to}} T'\}$ of $\bar{T}_a$ by $\bar{P}$. This implies in particular that $\bar{T}_a \overset{*}{\underset{\bar{P}}{\to}} T_a$, hence $T \overset{*}{\underset{\bar{P}}{\leftrightarrow}} \bar{T}_a$; it is easily seen that $\bar{T}_a = \bar{T}$, and since by Lemma 9, $P \models T \equiv T'$ implies $\bar{T} = \bar{T}'$, we have $\bar{T} \overset{*}{\underset{\bar{P}}{\to}} T'$ thus $T \overset{*}{\underset{\bar{P}}{\leftrightarrow}} T'$ for all T' semantically equivalent to T.

The proof that $P \models T_a \equiv T'$ implies $\bar{T}_a \overset{*}{\underset{\bar{P}}{\to}} T'$ is by induction on the structure of T': If $T' = X$, then $\bar{T}_a = T_a = X$ and indeed $X \overset{*}{\underset{\bar{P}}{\to}} X$. If $T' = F_i(T'_i)$ then $\bar{T}_a$ must be of the form $\bar{T}_a = F_i(\bar{T}'_i)$, otherwise it would not be reduced. By induction, $\bar{T}'_i \overset{*}{\underset{\bar{P}}{\to}} T'_i$ hence $\bar{T}_a \overset{*}{\underset{\bar{P}}{\to}} T'$. If $T' = G(T'_1,\ldots,T'_p)$ then either $\bar{T}_a = G(\bar{T}'_1,\ldots,\bar{T}'_p)$ or $\bar{T}_a = F_i(T'_a)$.

If $\bar{T}_a = G(\bar{T}'_1,\ldots,\bar{T}'_p)$ the argument proceeds nicely by induction. If $\bar{T}_a = F_i(T'_a)$, let $P_i = G(M_1,\ldots,M_p)$ be defining right-hand-side of $\bar{T}_a$ in $\bar{P}$. By definition of $\overset{*}{\underset{\bar{P}}{\rightarrow}}$, we have $\bar{T}_a \overset{*}{\underset{\bar{P}}{\rightarrow}} G(M_1\{T'_a\},\ldots,M_p\{T'_a\})$.

Since $M_i\{T'_a\}$ is reduced and $\bar{P} \models M_i\{T'_a\} \equiv T'_i$ for $1 \leq i \leq p$, we have $M_i\{T'_a\} = \bar{T}'_i$ by Lemma 9, hence $M_i\{T'_a\} \overset{*}{\underset{\bar{P}}{\rightarrow}} T'_i$ and $\bar{T}_a \overset{*}{\underset{\bar{P}}{\rightarrow}} T'$ follows by definition of $\overset{*}{\underset{\bar{P}}{\rightarrow}}$ again. □

For example, let $P = \langle F(X) \Leftarrow g(X,k(g(X,k(F(X),F(X))),F(X)))\rangle$, $T_1 = F(X)$ and $T_2 = g(X,k(F(X),F(X)))$. It is easily seen that the number of occurrences of the letter F in a term T_3 such that $T_1 \overset{*}{\underset{P}{\rightarrow}} T_3$ is odd, while it is even in a term T_4 such that $T_2 \overset{*}{\underset{P}{\rightarrow}} T_4$. It follows that T_1 and T_2 are <u>not</u> interconvertible with respect to P and yet $P \models T_1 \equiv T_2$. If we now consider $\bar{P} = \langle F_1(X) \Leftarrow g(X,F_2(X)),\ F_2(X) \Leftarrow k(F_1(X),F_1(X))\rangle$ and $T'_1 = F_1(X)$, $T'_2 = g(X,k(F_1(X),F_1(X)))$, it is easily checked that $\bar{T}'_1 = \bar{T}'_2 = F_1(X)$ and $F_1(X) \overset{*}{\underset{\bar{P}}{\rightarrow}} T'_1$, $F_1(X) \overset{*}{\underset{\bar{P}}{\rightarrow}} T'_2$ thus T'_1 and T'_2 are interconvertible with respect to $\bar{P}$.

5. A Formal Proof System and Its Completeness

In this section, we describe a formal system for proving properties of programs in our language and prove its completeness. The system is a straightforward adaptation of [2] or [18].

5.1 Description of the System

<u>Terms</u> are the same as terms of the programming language, with the addition that we can take the greatest-lower bound (g.l.b.) $\underline{\min}(T_1,T_2)$ of any two terms T_1 and T_2.

A <u>well formed formula</u> (wff) is a conjunction of inequalities $T \subseteq T'$

between terms; we use $T \equiv T'$ as an abbreviation for the conjunct $T \subseteq T'$, $T' \subseteq T$.

An assertion A is an expression $P,\ \Phi \vdash \Psi$ where Φ and Ψ are wffs and P is a program, i.e., a set of defining equations of the form $F_i(X) \Leftarrow P_i$, where P_i is a term expressed without min. We assume that each F_i is defined at most once in P. If $F_i(X) \Leftarrow P_i$ occurs in an assertion A, we say that F is bound in A; variable F_i is free in A if it is not bound. We assume the variable $\perp$ to be always bound by $\perp(X) \Leftarrow \perp(X)$.

A proof of an assertion A is a sequence of assertions $A_0,\ldots,A_n$ where $A_n = A$ and each assertion is derived from the previous ones by application of the axioms or induction rules.

Axioms

(Reflexivity)	A1:	$\vdash T \subseteq T$
(Transitivity)	A2:	$T_1 \subseteq T_2,\ T_2 \subseteq T_3 \vdash T_1 \subseteq T_3$
(Minimality)	A3:	$\perp(X) \Leftarrow \perp(X) \vdash \perp(T) \subseteq T',\ T\{\perp\} \subseteq T'$
(Monotonicity)	A4:	$T_1 \subseteq T_2,\ T_3 \subseteq T_4 \vdash T_1\{T_3\} \subseteq T_2\{T_4\}$
(g.l.b.)	A5:	$T \subseteq T_1,\ T \subseteq T_2 \vdash T \subseteq \underline{\min}(T_1,T_2)$
	A6:	$\vdash \underline{\min}(T_1,T_2) \subseteq T_1$
	A7:	$\vdash \underline{\min}(T_1,T_2) \subseteq T_2$
(fixed-point)	A8:	$F(X) \Leftarrow P \vdash T \equiv T\{\{P/F\}\}$

Here, T, T_1, T_2, T_3, T_4 and P denote arbitrary terms. A discussion regarding the introduction of min into the system can be found in [31].

Rules of Inference

They include the propositional calculus rules:

(Inclusion) R1: $\dfrac{P,\Phi\vdash\Psi}{P',\Phi'\vdash\Psi'}$ if $P\subseteq P'$, $\Phi\subseteq\Phi'$ and $\Psi'\subseteq\Psi$ (here $\subseteq$ is the set theoretic inclusion).

(Conjunction) R2: $\dfrac{P,\Phi\vdash\Psi \quad P,\Phi'\vdash\Psi'}{P,\Phi,\Phi'\vdash\Psi,\Psi'}$

(Cut) R3: $\dfrac{P,\Phi\vdash\Psi \quad P,\Psi\vdash\chi}{P,\Phi\vdash\chi}$

as well as Scott's induction rule:

(Induction) R4: $\dfrac{\Phi\vdash\Psi\{\bot(X)/F\} \quad \Phi,\Psi\vdash\Psi\{P/F\}}{F(X)\Leftarrow P,\Phi\vdash\Psi}$ (F free in $\Phi\vdash\Psi$)

Here, Φ, Φ', Ψ, Ψ' and χ denote arbitrary wffs, P and P' programs. By $\Psi\{P/F\}$, we mean the wff obtained from Ψ by substituting $T\{P/F\}$ for each term T which occurs in Ψ.

Although they can be derived from the previous ones, it will be convenient to also introduce the rules:

(Renaming) R5: $\dfrac{A}{A'}$ if A' is obtained from A by renaming some of the F variables.

(Subsumption) R6: $\dfrac{P,P',\Phi\vdash\Psi}{P,\Phi\vdash\Psi}$ if none of the variables bound by P' occurs free in $\Phi\vdash\Psi$.

(Parallel induction) R7: $\dfrac{P',\Phi\vdash\Psi\{\bot/F_1,\ldots,\bot/F_k\} \quad P',\Phi,\Psi\vdash\Psi\{P_1/F_1,\ldots,P_k/F_k\}}{F_1(X)\Leftarrow P_1,\ldots,F_k(X)\Leftarrow P_k,P',\Phi\vdash\Psi}$

provided each F_i is free in $P',\Phi\vdash\Psi$ for $1\leq i\leq k$.

A theorem is a provable assertion. Here are some examples of theorems easily proved within the system:

(i) $F(X) \Leftarrow T_1\{T_2\}$, $F'(X) \Leftarrow T_2'\{T_1'\} \vdash F(X) \equiv T_1'$, $F'(X) \equiv T_2$
where $T_1' = T_1\{F'/F\}$ and $T_2' = T_2\{F'/F\}$;

(ii) $F(X) \Leftarrow F(G(X)) \vdash F(X) \equiv \perp(X)$;

(iii) $F_1(X) \Leftarrow G(X,F_1F_1(X))$, $F_2(X) \Leftarrow G(X,F_1F_2(X)) \vdash F_1(X) \equiv F_2(X)$.

It is not quite so easy to find a proof of

(iv) $P \vdash F(X) \subseteq F_o(X)$ where

$$P = \begin{cases} F(X) \Leftarrow g(X,FFF(X),FFF(X)) \\ F_o(X) \Leftarrow g(X,F_1F_oF_1(X),F_1F_2(X)) \\ F_1(X) \Leftarrow g(X,F_oF_2(X),F_2F_1(X)) \\ F_2(X) \Leftarrow g(F_o(X),F_2F_2(X),F_oF_2F_1(X)) \end{cases}$$

The proof of (iv) generated by the method of Theorem 3 (below) uses Scott's Induction, on assertion

$$P' \vdash F(X) \subseteq F_o(X), \quad F(X) \subseteq F_1(X), \quad F(\underline{\min}(F_o(X),F_1(X))) \subseteq F_2(X) \quad ,$$

where P' is obtained from P by removing the defining equation for F.

It is not known to the authors whether this particular proof (or any other for that matter) indeed requires the use of min.

Completeness of the Formal System

We now prove the main result of this section:

Theorem 3: The formal system described above is (effectively) complete with respect to acceptable program P, i.e., $P \models T \equiv T'$ if and only if $P \vdash T \equiv T'$.

Proof. Just for convenience, we restrict ourselves to acceptable programs in standard form, since there is no difficulty in extending the result to all acceptable programs. The direction $P \vdash T \equiv T'$ implies $P \models T \equiv T'$ expresses the soundness of the system which is shown in [18] or [31]. In order to prove that $P \models T \equiv T'$ implies $P \vdash T \equiv T'$, we notice that, using only axiom A8, it is easy to prove that $T \overset{*}{\underset{P}{\leftrightarrow}} T'$ implies $P \vdash T \equiv T'$.

By Theorem 2, we know that $\bar{P} \models T \equiv T'$ if and only if $T \overset{*}{\underset{\bar{P}}{\leftrightarrow}} T'$.

It follows that $\bar{P} \models T \equiv T'$ if and only if $\bar{P} \vdash T \equiv T'$ and Theorem 3 is proved for programs $\bar{P}$ in canonical form. Let $P \models \langle F_1 \Leftarrow P_1, \ldots, F_m \Leftarrow P_m \rangle$ be an arbitrary program in standard form and $\bar{P} = \langle \bar{F}_1 \Leftarrow \bar{P}_1, \ldots, \bar{F}_m \Leftarrow \bar{P}_m \rangle$ its canonical representation. In order to prove that $P \models T \equiv T'$ implies $P \vdash T \equiv T'$, it is sufficient to prove that

$$P, \bar{P} \vdash \bar{F}_1 \equiv F_1, \ldots, \bar{F}_m \equiv F_m \quad . \tag{a}$$

We know that $P \models T \equiv T'$ implies $\bar{P} \models \bar{T} \equiv \bar{T}'$ where $\bar{T} = T\{\bar{F}_1/F_1, \ldots, \bar{F}_m/F_m\}$ and $\bar{P} \models \bar{T} \equiv \bar{T}'$ implies $\bar{P} \vdash \bar{T} \equiv \bar{T}'$; formula (a) implies $P, \bar{P} \vdash T \equiv \bar{T}$, $T' \equiv \bar{T}'$ thus $P \models T \equiv T'$ implies $P \vdash T \equiv T'$.

It is easy to show that $P, \bar{P} \vdash F_1 \subseteq \bar{F}_1, \ldots, F_m \subseteq \bar{F}_m$ and we leave this to the reader. In order to prove the other way around, i.e., $P, \bar{P} \vdash \bar{F}_1 \subseteq F_1, \ldots, \bar{F}_m \subseteq F_m$, we introduce the following notation: let $\sum(M)$ represent the greatest lower bound of the finite set $\{M' \mid P \models M' \equiv M\}$ of terms equivalent to M. For example, if $P \models F_1 \equiv F_2F_2$ is the only valid elementary formula, then $\sum(F_1F_2) = \underline{\min}(F_1F_2, F_2F_2F_2, F_2F_1)$ which is just an abbreviation for $\underline{\min}(F_1F_2, \underline{\min}(F_2F_2F_2, F_2F_1))$.

For any $1 \leq i \leq m$ there is a unique valid formula $P \models F_i \equiv F_{j_1} \cdots F_{j_k}$

where $\bar{F}_{j_1},\ldots,\bar{F}_{j_k}$ are active variables in $\bar{P}$. Let w_i represent the wff $\bar{F}_{j_1}(\sum(F_{j_2}\cdots F_{j_k})) \subseteq F_i$ and Φ be the conjunction $\bigwedge_{1\leq i\leq m} w_i$ of all w_i's. In order to prove (a), it is sufficient to prove

$$P, \bar{P} \vdash \Phi \quad . \tag{b}$$

If $\bar{F}_i$ is active in $\bar{P}$, then w_i is $\bar{F}_i \subseteq F_i$ and, since we know $P, \bar{P} \vdash F_i \subseteq \bar{F}_i$, proving (b) implies $P, \bar{P} \vdash \bar{F}_i \equiv F_i$ for all F_i's which are active in $\bar{P}$.

If $\bar{F}_i$ is not active in $\bar{P}$ then $\bar{P} \vdash \bar{F}_i \equiv \bar{F}_{j_1}\cdots\bar{F}_{j_k}$ by A8, and $\bar{F}_{j_1},\ldots,\bar{F}_{j_k}$ are active in $\bar{P}$; it follows from the remark that anything which is true in $\bar{P}$ is provable that $\bar{P} \vdash \bar{F}_{j_2}\cdots\bar{F}_k \subseteq \bar{N}$ for any $\bar{N} \in \sum(\bar{F}_{j_2}\cdots\bar{F}_{j_k})$, hence, by repeated use of A5: $\bar{P} \vdash \bar{F}_{j_2}\cdots\bar{F}_k \subseteq \sum(\bar{F}_{j_2}\cdots\bar{F}_{j_k})$.

Since we already know that $P, \bar{P} \vdash \bar{F}_{j_2} \equiv F_{j_2},\ldots,\bar{F}_{j_k} \equiv F_{j_k}$ we obtain, by A2 and A4 $P, \bar{P} \vdash \bar{F}_i \subseteq \bar{F}_{j_1}\sum(F_{j_2}\cdots F_{j_k})$. Assuming (b) now implies $P, \bar{P} \vdash w_i$, i.e., $P, \bar{P} \vdash \bar{F}_{j_1}\sum(F_{j_2}\cdots F_{j_k}) \subseteq F_i$ hence, by A2 again $\bar{P}, P \vdash \bar{F}_i \subseteq F_i$ for all $1 \leq i \leq m$.

Before proving (b), we need to establish the following

Lemma: $P \models M_1M_2 \equiv M$ and all variables in $\bar{M}_1$ active in $\bar{P}$ implies $\Phi \vdash \bar{M}_1\{\sum(M_2)\} \subseteq M$.

Proof. The proof is by induction on $\delta(M)$.

If $\delta(M) = 0$ then $M = M_1 = M_2 = X$ and $\Phi \vdash X \subseteq X$ by A1.

If $\delta(M) > 0$ then $M = F_i(M')$. Let $P \models F_i \equiv F_{j_1}\cdots F_{j_k}$ with $\bar{F}_{j_1},\ldots,\bar{F}_{j_k}$ active in $\bar{P}$. We consider two cases:

Case 1: $\delta(M_1) \geq \delta(F_i)$

In this case $\bar{M}_1 = \bar{F}_{j_1}\cdots\bar{F}_{j_k}\bar{M}_1'$ and we must prove $\Phi \vdash \bar{F}_{j_1}\cdots\bar{F}_{j_k}\{\bar{M}_1'\{\sum(M_2)\}\} \subseteq F_i(M')$.

Since $\delta(F_{j_2}\cdots F_{j_k}) < \delta(F_i) \leq \delta(M)$, we know by induction that $\Phi \vdash \bar{F}_{j_2}\cdots\bar{F}_{j_k} \subseteq N$ for any $N \in \sum(F_{j_2}\cdots F_{j_k})$. Hence, by repeated applications of A5, we obtain $\Phi \vdash \bar{F}_{j_2}\cdots\bar{F}_{j_k} \subseteq \sum(F_{j_2}\cdots F_{j_k})$, hence, by A4 $\Phi \vdash \bar{F}_{j_1}\cdots\bar{F}_{j_k} \subseteq \bar{F}_{j_1}(\sum(F_{j_2}\cdots F_{j_k}))$. Since $\bar{F}_{j_1}(\sum(F_{j_2}\cdots F_{j_k})) \subseteq F_i$ belongs to Φ by construction, an application of A2 gives us $\Phi \vdash \bar{F}_{j_1}\cdots\bar{F}_{j_k} \subseteq F_i$.

By induction again, we know that $\Phi \vdash \bar{M}_1'\{\sum(M_2)\} \subseteq M'$ and one application of A4 allows us to conclude $\Phi \vdash \bar{M}_1\{\sum(M_2)\} \subseteq M$.

Case 2: $\delta(M_1) < \delta(F_i)$

Then $\bar{M}_1 = \bar{F}_{j_1}\cdots\bar{F}_{j_k}$ with $n < k$ and $P \models M_2 \equiv F_{j_{n+1}}\cdots F_{j_k}M'$.

It is clear by definition of $\sum$ and repeated applications of A6 and A7 that $\vdash \sum(M_2) \subseteq \sum(F_{j_{n+1}}\cdots F_{j_k})\{\sum(M')\}$ hence, by A4 $\Phi \vdash \bar{M}_1\{\sum(M_2)\} \subseteq \bar{M}_1\{\sum(F_{j_{n+1}}\cdots F_{j_k})\{\sum(M')\}\}$.

By induction, $\Phi \vdash \bar{F}_{j_2}\cdots\bar{F}_{j_n}\sum(F_{j_{n+1}}\cdots F_{j_k}) \subseteq N$ for any $N \in \sum(F_{j_2}\cdots F_{j_k})$. Hence, by repeated applications of A5: $\Phi \vdash \bar{F}_{j_2}\cdots\bar{F}_{j_n}\{\sum(F_{j_{n+1}}\cdots F_{j_k})\} \subseteq \sum(F_{j_2}\cdots F_{j_k})$. Using the same proof as in case 1, we then obtain $\Phi \vdash \bar{M}_1\{\sum(F_{j_{n+1}}\cdots F_{j_k})\} \subseteq F_i$. Since $\vdash \sum(M') \subseteq M'$ by A6 and A7, we can combine everything into $\Phi \vdash \bar{M}_1\{\sum(M_2)\} \subseteq F_i(M')$. □

We are now ready to prove (b), i.e., $P, \bar{P} \vdash \Phi$ by parallel Scott induction R7 on $P \vdash \Phi$.

Basis: We must prove $P \vdash \Phi\{\bot/\bar{F}_j\}$ for all $\bar{F}_j$ active in $\bar{P}$. This is easy using A3.

Induction: Assuming P, Φ, we must prove $\Phi\{\bar{P}_j/\bar{F}_j\}$ for all $\bar{F}_j$ active in $\bar{P}$, i.e., that $\vdash \bar{P}_{j_1}\{\sum(F_{j_2}\cdots F_{j_k})\} \subseteq F_i$ for all $1 \leq i \leq m$. Let $P_i = g(M_1,\ldots,M_p)$; using A8, we show $P \vdash P_i \equiv F_i$ thus it will be sufficient to prove that $\Phi \vdash \bar{P}_{j_1}\{\sum(F_{j_2}\cdots F_{j_k})\} \subseteq P_i$. The term P_{j_i} can only be of the form $\bar{P}_{j_1} = g(\bar{M}_1,\ldots,\bar{M}_p)$ and thus, all we need to prove is $\Phi \vdash \bar{M}_k\{\sum(F_{j_2}\cdots F_{j_k})\} \subseteq M_k$ for all $1 \leq k \leq p$. Since we know from

the preceding Lemma that this is the case, we can regroup everything using A4 and the proof of Theorem 3 is completed. □

6. Conclusion

There are many directions in which this type of study could be generalized. We simply mention a few outstanding unanswered questions:

1. Are the main results obtained here still valid without the restriction that $\delta(F_i(X)) \neq \infty$?

In terms of languages, this means being able to decide the equivalence of simple language over infinite as well as finite words. The corresponding problem for regular languages is solved in [7].

2. Is equivalence between polyadic programs decidable?

It is shown by Courcelle in [5] that equivalence between polyadic recursive schemes is decidable if and only if equivalence between deterministic pushdown automatons is also decidable.

3. Do similar results exist about the typed λ-calculus and λ-calculus? Is Milner's LCF [17] complete if we remove conditionals?

4. Can one prove Theorem 3 without introducing min or, (as the authors conjecture), show that min is necessary?

Acknowledgments

We wish to express our gratitude to Gilles Kahn for his collaboration to this work, and to Robin Milner and Maurice Nivat for many stimulating discussions on the subject. J.W. deBakker pointed out an error in an earlier manuscript.

References

[1] E.A. Ashcroft, Z. Manna, A. Pnueli, "Decidable Properties of Monadic Functional Schemas," in Theory of Machines and Computations, Kohavi and Paz, eds. Academic Press ( ), pp. 3-18.

[2] J.W. deBakker, "Recursive Procedures," Mathematical Center Tracts No. 24, Amsterdam (1971).

[3] J.W. deBakker, W.P. deRoever, "A Calculus for Recursive Program Schemes," Proceedings of IRIA Colloquium, North-Holland (1972).

[4] J.W. deBakker, D. Scott, "A Theory of Programs," unpublished (1969).

[5] B. Courcelle, "Recursive Schemes, Algebraic Trees, Deterministic Languages," Proceedings of the 15th SWAT Symposium (1974).

[6] B. Courcelle, "Grammaires Canoniques des Langages Simples Déterministes," RAIRO No. 1, Paris (1974).

[7] B. Courcelle, G. Kahn, J. Vuillemin, "Algorithmes d'Equivalence pour des Equations Récursives Simples," Rapport Laboria No. 37, IRIA (1973).

[8] B. Courcelle, M. Nivat, J. Vuillemin, "Recursive Schemes," Rapport Laboria, IRIA (to appear).

[9] B. Courcelle, J. Vuillemin, "Complètude d'un système formel pour l'Equivalence de Schèmas Récursifs monadiques," Comptes-Rendu du Collogue de Paris, Institut de Programmation (April 1974).

[10] M. Gordon, "Evaluation and Denotation of Pure LISP Programs: a Worked Example in Semantics," Thesis, Edinburgh (1973).

[11] M.S. Paterson, C. Hewitt, "Comparative Schematology," in Record of Project MAC Conference on Parallel Computation, ACM, New York (1970).

[12] P. Hitchcock, D. Park, "Induction Rules and Proofs of Termination," Proceedings of IRIA Colloquium, North-Holland (1972).

[13] J.E. Hopcroft, A.J. Korenjak, "Simple Deterministic Languages," 7th SWAT Symposium (1966), pp. 36-46.

[14] Y.I. Ianov, "The Logical Schemes of Algorithms," Problems in Cybernetics, Vol. 1, Pergamon Press (1960).

[15] G. Kahn, "A Preliminary Theory of Parallel Programs," Rapport Laboria No. 6, IRIA (1973).

[16] D. Luckham, D. Park, M.S. Paterson, "On Formalized Computer Programs," JCSS, Vol. 4, No. 3 (1970), pp. 220-249.

[17] Z. Manna, S. Ness, J. Vuillemin, "Inductive Methods for Proving Properties of Programs," CACM, Vol. 16, No. 8 (1973).

[18] R. Milner, "Models of LCF," AIM-186/CS-332, Stanford University (1973).

[19] R. Milner, "Processes: a Model of Computing Agents," University of Edinburgh Memorandum (1973).

[20] R. Milner, R. Weyrauch, "Proving Compiler Correctness in a Mechanized Logic," Machine Intelligence 7, Edinburgh University Press (1972).

[21] M. Nivat, "On the Interpretation of Polyadic Recursive Schemes," Rapport Laboria, IRIA (1974).

[22] M. Paterson, "Program Schemata," in Machine Intelligence 3, Edinburgh University Press (1968).

[23] G. Plotkin, "LCF Considered as a Programming Language," School of Artificial Intelligence, University of Edinburgh Memorandum (1974).

[24] W.P. deRoever, "Operational, Mathematical and Axiomatized Semantics for Recursive Procedures and Data Structures," Mathematisch Centrum Report, Amsterdam (1974).

[25] B. Rosen, "Program Equivalence and Context-Free Grammars," IBM Research Memo, Yorktown Heights (1974).

[26] D. Scott, "Outline of a Mathematical Theory of Computation," Oxford Monograph PRG-2, Oxford University (1970).

[27] D. Scott, "Continuous Lattices," Oxford Monograph PRG-7, Oxford University (1972).

[28] D. Scott, C. Strachey, "Towards a Mathematical Semantics for Computer Languages," Oxford Monograph PRG-6, Oxford University (1972).

[29] J. Vuillemin, "Proof Techniques for Recursive Programs," Ph.D. Thesis, Stanford University (1973).

[30] J. Vuillemin, "Correct and Optimal Implementations of Recursion in a Simple Programming Language," Proceedings of the Fifth ACM Symposium on Theory of Computing (1973), pp. 244-239.

[31] J. Vuillemin, "Syntaxe, Sémantique et Axiomatique d'un Langage de Programmation Simple," Thèse d'État, Université de Paris VII (1974).