(needs to be rewritten)

On degrees of productivity

In this note I introduce some notions that might be useful for the discussion of the "productivity" of SASL programs. Whether these notions are sufficiently suited to be used in calculations remains to be investigated. This aspect, however, I consider a separable one. In this note only definitions and properties will be recorded; most of the proofs are straightforward and, therefore, omitted.

The universe of discourse is the collection of all infinite sequences of integers. We are interested in the computability of solutions of equations of the type $x = F.x$ where F is a function on our universe. It has been observed that the existence of a unique solution of such an equation does not imply the computability of that solution [0], where, of course, "computability" should be interpreted with a particular class of implementations in mind; in our case: implementations based on a reduction strategy. For example:

$x = 1 : x$ has a unique, computable solution, whereas
$x = twice \circ x$, where $twice.i = 2 * i$, has a unique

solution too, viz. the infinite sequence consisting of zeroes only, but, in concordance with the author of [0], "I expect no implementation to find it". Apparently, two types of functions exist, viz. those that yield "productive" equations and those that do not. The problem is how to characterise the productive ones.

Aside on notation: Functional application is denoted by the infix operator $\cdot$ ("dot"); it is left associative and it has the largest binding power. Functional composition is denoted by the infix operator $\circ$ ("after"); it is associative and it has the second largest binding power. Whenever it is convenient a sequence will be interpreted as an integer function on the naturals. I.e: I permit myself to use $x.n$ to denote the n-th element $(0 \leq n)$ of the sequence x and to use $f \circ x$ for the sequence obtained by application of the integer function f to all elements of x. (I.e: $(f \circ x).n = f.(x.n)$, for all natural n). $f\circ$ is the corresponding function on sequences.

For any integer b a function $b:$ on sequences exist, defined by:

$$(b:x).0 = b,$$
$$(b:x).(n+1) = x.n \text{ , for all natural } n.$$

The functions id and tl on sequences are defined by:

$$id.x = x,$$
$$tl.x.n = x.(n+1) \text{ , for all natural } n.$$

(end of aside on notation).

In the sequel, f and g denote integer functions, F and G denote functions on sequences, and x and y denote sequences.

Def0: for any integer n the equivalence relation ν_n on sequences is defined by:

$$x \,\nu_n\, y \equiv (\underline{A}i: 0 \leq i < n: x.i = y.i).$$

Def1: a function F is called "n-productive" if:
$(\underline{A}x,y :: (\underline{A}k :: x \,\nu_k\, y \Rightarrow F.x \,\nu_{n+k}\, F.y))$, for integer n.

Lemma 0: id is 0-productive.
Lemma 1: b: is 1-productive, for any integer b.
Lemma 2: tl is (-1)-productive.
Lemma 3: f∘ is 0-productive, for any f.
Lemma 4: for any F and any integer n:
F is (n+1)-productive ⇒ F is n-productive.

Not any function is n-productive for some n:
Example 0: the function half, defined by:
half.(a:b:x) = a:half.x, is not n-productive, for any n. (end of example).

Lemma 5: F is m-productive ∧ G is n-productive
⇒ F∘G is (m+n)-productive.

Def 2: F is "positively productive" (or "productive", for short) is: (∃n: 1≤n: F is n-productive).

Lemma 6: F is productive ≡ F is 1-productive.

The following theorem makes all the above troubles worthwhile:

Theorem 0: "the equation x = F.x has a computable solution"
⇐ "F is productive".

proof: Assuming that F is productive and, of course, that F itself is computable (what this should mean is discussed after completion of the proof), we derive an algorithm for the computation of a solution of x = F.x.

The algorithm computes the elements x.i of the solution in the order of increasing i. Assuming that, for n: 0≤n, the elements x.i, 0≤i<n, have been

computed, $x.n$ can be computed in the following way: Select a sequence y satisfying: $(\underline{A}i : 0 \leq i < n : y.i = x.i)$. Then, let $x.n = F.y.n$. Because F is computable, the computation of $x.n$ takes only a finite amount of time. Moreover, if z is a sequence satisfying: $(\underline{A}i : 0 \leq i < n : z.i = x.i)$ then:

$$
\begin{array}{ll}
 & (\underline{A}i : 0 \leq i < n : z.i = x.i) \wedge (\underline{A}i : 0 \leq i < n : y.i = x.i) \\
\Rightarrow & \{ \text{transitivity of } = \} \\
 & (\underline{A}i : 0 \leq i < n : z.i = y.i) \\
= & \{ \text{definition of } \nu_n \} \\
 & z \; \nu_n \; y \\
\Rightarrow & \{ F \text{ is 1-productive} \} \\
 & F.z \; \nu_{n+1} \; F.y \\
= & \{ \text{definition of } \nu_{n+1} \} \\
 & (\underline{A}i : 0 \leq i < n+1 : F.z.i = F.y.i) \\
\Rightarrow & \{ \text{calculus} \} \\
 & F.z.n = F.y.n \; .
\end{array}
$$

From this derivation we conclude that the value $x.n$ is unique. We now have proved that if $x.i$, $0 \leq i < n$, is computable in a finite amount of time then $x.n$ is computable in a finite amount of time. The proof that the sequence x thus constructed indeed is a solution of the equation is left to the reader.
(end of theorem).

Def 3: "A function F on sequences is computable" $\equiv$ "for any sequence x and any natural n: $F.x.n$ can be computed in finitely many steps involving finitely many elements of x only".

The fact that I initially forgot to include Def3 probably is due to the fact that I directed all my attention to the productivity of equations, thereby taking the computability of the functions involved for granted.

The theorem shows that 1-productivity is the key-notion. The generalisation to n-productivity, even for negative n, has been made to enable easy calculation of the degree of productivity for the composition of functions.

Example 1: 1: is 1-productive (Lemma 1), twice ∘ is 0-productive (Lemma 3), hence F, where F.x = 1: twice∘x, is 1-productive (Lemma 5), hence the equation

x = 1: twice∘x is productive. (end of example).

(Note: in this example, I call an equation productive if on account of Theorem 0 its solution is computable).

Example 2: By suitable definition of n-productivity of functions of more than 1 arguments it is possible to derive: "merge is 0-productive" ⇒

"F, where F.x = 1: merge.(twice∘x).(thrice∘x), is productive"; hence, the equation

x = 1: merge.(twice∘x).(thrice∘x) is productive.

(end of example).

1985.10.3
Rob Hoogerwoord

[0]: "Hamming's exercise in SASL", Edsger W. Dijkstra. (EWD792).