J. McCarthy

AN INTERESTING LISP FUNCTION

Ikuo Takeuchi (1978) of the Electrical Communication Laboratory of Nippon Telephone and Telegraph Co. (Japan's Bell Labs) devised a recursive function program for comparing the speeds of LISP systems. It can be made to run a long time without generating large numbers or using much stack. The program is

$$\begin{aligned} tak(x,y,z) \leftarrow\ & \textbf{if } x \leq y \textbf{ then } y \\ & \textbf{else } tak(tak(x-1,y,z), tak(y-1,z,x), tak(z-1,x,y)), \end{aligned}$$

where the variables may range over the integers (including negative) or else over real numbers. The program has similar properties in the two cases, but proving termination seems more tedious if arbitrary real numbers are allowed. The program has several interesting features.

1. Inspection suggests that *tak* satisfies the equation

$$tak(x+a, y+a, z+a) = tak(x,y,z) + a,$$

whenever the computation terminates, and this can be shown by subgoal induction. Namely, it is true for the non-recursive case, and assuming it for the referred sets of arguments yields it for the main set. A formal proof can proceed along the lines suggested in (McCarthy 1978).

2. Experiment using LISP indicates first of all that the value of $tak(x,y,z)$ is always one of x, y or z. I don't see a proof of this fact in isolation.

3. Like the 91-function, the program computes a simple non-recursive function. Using LISP to compute some values of *tak* leads to the guess that it is the same as

$$tak0(x,y,z) = \textbf{ if } x \leq y \textbf{ then } y \textbf{ else if } y \leq z \textbf{ then } z \textbf{ else } x.$$

Substitution shows that *tak*0 satisfies the functional equation for *tak*, and therefore by the *minimization schema* of (McCarthy 1978),

$$tak(x,y,z) = tak0(x,y,z)$$

whenever the former terminates. *A fortiori*, this establishes that $tak(x,y,z)$ takes one of the variables as value, but maybe that fact could be proved separately.

4. In order to prove that *tak* is total, we write a *"derived program"* $dtak(x,y,z)$ that computes the depth of recursion involved in computing $tak(x,y,z)$ using call-

by-value. We have

$$
\begin{aligned}
dtak(x,y,z) \leftarrow\ & \textbf{if } x \le y \textbf{ then } 0 \\
& \textbf{else } 1 + \max(dtak(x-1,y,z), \\
& \qquad dtak(y-1,x,z), \\
& \qquad dtak(z-1,x,y), \\
& \qquad dtak(tak(x-1,y,z), tak(y-1,z,x), tak(z-1,x,y))).
\end{aligned}
$$

Experiment with LISP leads to the conjecture that for integer values of the variables, *dtak* is extensionally equivalent to

$$dtak0(x,y,z) = dtak00(x-y, z-y),$$

where

$$
\begin{aligned}
dtak00(m,n) =\ & \textbf{if } m \le 0 \textbf{ then } 0 \\
& \textbf{else if } n \ge 2 \textbf{ then } m + n(n-1)/2 - 1 \\
& \textbf{else if } n \ge 0 \textbf{ then } m \\
& \textbf{else if } n = -1 \textbf{ then } (m+1)(m+2)/2 - 1 \\
& \textbf{else } (m-n)(m-n+1)/2 - m - 1.
\end{aligned}
$$

We don't bother to verify the conjecture. Instead we use *dtak*0 as a rank function to prove $\forall i.\Phi(i)$ by course-of-values induction where

$$\Phi(i) \equiv \forall xyz.(dtak0(x,y,z) = i \supset tak(x,y,z) = tak0(x,y,z)).$$

Since we already know that *tak*0 satisfies the functional equation of *tak*, we need only show that in the recursive case of *tak*, i.e. when $x > y$, the referred arguments are of lower rank than the main ones. Thus we must show that each of $dtak0(x-1,y,z)$, $dtak0(y-1,z,x)$, $dtak0(z-1,x,y)$, and $dtak0(tak0(x-1,y,z), tak0(y-1,z,x), tak0(z-1,x,y)$ is strictly less than $dtak0(x,y,z)$. Each inequality follows from a separate easy case analysis. Presumably termination could be proved for the non-integer case by a similar argument with a more complicated formula for *dtak*00. We leave this as an exercise for the reader.

5. Like Morris's program

$$morris(x,y) \leftarrow \textbf{ if } x = 0 \textbf{ then } 1 \textbf{ else } morris(x-1, morris(x,y)),$$

tak is more quickly computed by call-by-need (Vuillemin 1973). In fact the number of function calls appears to be exponential with call-by-value and quadratic with call-by-need. The culprit is the third argument of the outer call, namely $tak(z-1,x,y)$ which is often unneeded. Unlike *morris*, however, *tak* is total.

A call-by-need version of *tak* is given by

$$ntak(x, y, z) \leftarrow vtak\langle x, y, z\rangle,$$

where

$$\begin{aligned} vtak\, u \leftarrow\ & \textbf{if}\ numberp\, u\ \textbf{then}\ u \\ & \textbf{else if n d}\ u\ \textbf{then}\ vtak(\textbf{ a }u) - 1 \\ & \textbf{else}\ \{vtak\ \textbf{a}\ u, vtak\ \textbf{ad}\ u\}[\lambda xy. \\ & \quad \textbf{if}\ x \leq y\ \textbf{then}\ y \\ & \quad \textbf{else}\ vtak\langle\langle x-1, y,\ \textbf{add}\ u\rangle, \langle y-1,\ \textbf{add}\ u, x\rangle, \langle\ \textbf{add}\ u - 1, x, y\rangle\rangle]. \end{aligned}$$

vtak is obtained from *tak* in a somewhat ad hoc way. Since we don't always compute all arguments of a function we must work with triples whose elements are either numbers or applications of functions to arguments. The only functions that occur are *vtak* itself and subtracting one. Therefore, a number stands for itself, an application of *vtak* is represented by a triple, and an application of subtracting one is represented by a list of one element—the inner expression from which one is to be subtracted. Perhaps I'm being thick, but it seems to me that call-by-need requires lists or at least putting symbols on the stack.

MACLISP versions of *tak* and friends are in the file TAK.LSP[F78,JMC] at SU-AI. I thank Don Knuth for suggesting call-by-need. The external or publication LISP notation is as in (McCarthy and Talcott 1978). It has the following features: (1) **a** and **d** are used for *car* and *cdr*, and their compounds are formed similarly. (2) The infix . is used for *cons*, and $\langle x, y, \ldots, z\rangle$ is used for $list[x, y, \ldots, z]$. (3) $\{x, y, \ldots, z\}f$ is used for $f[x, y, \ldots, z]$ whenever we prefer to write the arguments first and the function name later.

References

(McCarthy 1978) John McCarthy, Representation of recursive programs in first order logic, *Proceedings of the International Conference on Mathematical Studies of Information Processing,* Kyoto, August 1978, 587–605.

(McCarthy and Talcott 1978) John McCarthy and Carolyn Talcott, **LISP: Programming and Proving**, preliminary edition, Computer Science Department, Stanford University, 1978.

(Takeuchi 1978) Ikuo Takeuchi, personal communication.

(Vuillemin 1973) Jean Étienne Vuillemin, *Proof techniques for recursive programs,* Ph.D. thesis, Computer Science Department, Stanford University, 1973.