The equivalence of certain computations

By D. C. Cooper*

Both iterative and recursive programs for computing generalizations of functions which obtain the factorial of an integer, which reverse the order of symbols on a list and which obtain the approximate integral of a function are given as recursive definitions using conditional expressions. The equivalence of the iterative and recursive definitions are proved and a general theorem on equivalence, from which some of the results may be deduced, is stated and proved.†

J. McCarthy (1960, 1962 and 1963) has laid a foundation for a mathematical theory of computation, a foundation that provides a mechanism for defining computable functions in a manner that is more natural than other schemes such as Turing machines and Markov algorithms. By more natural here we mean that the formalism is closer to the way we express the computable functions that concern us than are other schemes with the same amount of generality. Hopefully then the system is one in which it should be easier to prove theorems (as contrasted with proving metatheorems about the system) than it is in other foundations. It is the purpose of this paper to take some "real problems" and to see how the system deals with them.

The particular problem with which we shall be concerned is proving the equivalence of two computations. By saying that two computations are equivalent over some set of legal data for them we mean that if the same legal data is presented to the two programs defining these two computations then either both programs will fail to produce an answer (i.e. get stuck in a loop) or they will both produce the same answer. If the computations are represented by function definitions (as in McCarthy's theory of computation) then we take two functions as being equivalent over some domain of values if for any argument in that domain either both functions are undefined or both have the same value. We shall take some computations which we "know" are equivalent and then prove this equivalence. These computations will take the form of (*a*) recursive programs to compute some function and (*b*) equivalent iterative programs that compute the same function. For a certain class of recursively defined functions, essentially those which can be defined by the primitive recursion schema, we show some circumstances in which an automatic conversion to an iterative program could be made. Having produced these "known equivalent" computations we then prove their equivalence; from these proofs we spot a general theorem on the equivalence of two recursive definitions and then prove the theorem. "General" here means that the theorem can be applied in more than one case, rather than in all cases!

The computations we wish to prove equivalent will initially be in the form of a computer program (for example written in the ALGOL language) or perhaps in the form of a flow chart. These must be converted into recursive definitions of functions, the definitions normally involving conditional expressions. McCarthy (1962) gives two different methods for this process. In his Sections 12 and 13 he sketches a method for defining a recursive function *algol* (π, ξ) whose value is the state of the "algol machine" at the conclusion of the algol program π, when this program is started with the state of the machine ξ. The equivalence of two particular programs, λ and μ say, is then a property of the function *algol*, namely *algol* $(\lambda, \xi) =$ *algol* (μ, ξ), and to prove equivalence we must prove that this equation is satisfied for all relevant ξ, where *algol* is a given function. This "interpreter" approach seems well adapted to answering questions about compilers, but not so convenient for answering questions about particular programs. An alternative "compiler" approach is described in McCarthy's Section 9 and also in an earlier paper (1960). In this approach a mechanical procedure is described for producing from a flow chart an equivalent recursive function definition. Corresponding to any two programs, λ and μ represented by flow charts, we then have two recursive functions $f_\lambda(\xi)$ and $f_\mu(\xi)$, and to prove equivalence of the two programs we have to prove $f_\lambda(\xi) = f_\mu(\xi)$ for the relevant ξ. This is the approach we adopt in this paper, except that we shall take as arguments of functions those components of ξ that are relevant rather than have a single argument ξ.

Notation

This paper is based on the recursive definition of functions using conditional expressions, as introduced by McCarthy (1960, 1962, 1963). We use the notation

$$[P_1 \to e_1;\ P_2 \to e_2;\ \ldots;\ P_n \to e_n;;e_{n+1}]$$

which corresponds to the ALGOL expression

if P_1 **then** e_1 **else if** P_2 **then** e_2 . . . **else if** P_n **then** e_n **else** e_{n+1}

This notation seems to avoid confusion with round parentheses and commas which are only used in this

† The research reported here was supported by the Advanced Research Projects Agency of the Department of Defense under the Contract SD-146 to the Carnegie Institute of Technology.

* *Computer Science Department, Carnegie Institute of Technology, Pittsburgh, Pennsylvania, U.S.A.*

paper to enclose and separate arguments of functions. The final $;;e_{n+1}$ represents an "in all other cases" expression and need not be present. The most common conditional expression in this paper takes the form $[P \rightarrow x;;y]$ and is McCarthy's elementary conditional form (1963).

A device is used to aid readability by cutting down the number of brackets enclosing function arguments. A function with one argument, $F(x)$ say, may be written with the argument as a subscript on the function name, i.e. F_x. This device is not always used, we may still write $F(x)$, but subscripts are used for no other purpose in the remainder of this paper.

Example 1: the factorial function

We start by considering three ALGOL procedure declarations for the factorial function, the first a recursive definition and the other two iterative definitions.

```
integer procedure Fr(N); value N; integer N;
  Fr := if N = 0 then 1 else N*Fr(N − 1)
integer procedure Fd(N); value N; integer N;
  begin integer A; A:=1;
  L: if N > 0 then begin A := N*A; N := N − 1;
                                  go to L end; Fd := A
  end
integer procedure Fu(N); value N; integer N;
  begin integer A, M; A := 1; M := 0;
  L: if M < N then begin M := M + 1; A := A*M;
                                  go to L end; Fu := A
  end
```

Writing the above ALGOL definitions as recursive function definitions using conditional expressions we have:

$$Fr(N) = [N = 0 \rightarrow 1;;N*Fr(N - 1)]$$

$$Fd(N) = Gd(N, 1) \text{ where}$$

$$Gd(N, A) = [N = 0 \rightarrow A;;Gd(N - 1, N*A)]$$

$$Fu(N) = Gu(N, 0, 1) \text{ where}$$

$$Gu(N, M, A) = [M = N \rightarrow A;;Gu(N, M + 1, (M + 1)*A)].$$

The equivalence of $Fr(N)$, $Fd(N)$ and $Fu(N)$ may now be proved using recursion induction (see McCarthy (1963), page 58), properties of conditional expressions and properties of the functions occurring in the definitions, i.e. plus, minus and multiplication. Rather than do this we first obtain some function definitions of which these are special cases, and obtain proofs for these generalizations.

Exaınple 2: generalization of the factorial function

We are more interested in the form of the definition of Fr than in the fact that it involves the multiplication and predecessor functions; let us therefore try to generalize the definitions so as not to include specific functions nor depend on properties of the integers. In the definition of Fr then:

(i) Replace $N - 1$ by a general "decrementing" function $\delta(N)$.
(ii) Replace the multiplication by a general function H with two arguments.
(iii) Replace the numerical constants 0 and 1 by L and B.

There are then obvious similar replacements to be made in the definitions of Gd and Gu, but two further replacements are necessary:

(*iv*) In the definition of Gu we have "$M + 1$"; adding 1 is the inverse operation to subtracting 1 and therefore we must replace $M + 1$ by $\sigma(M)$ where σ and δ are related by

$$N = \sigma(\delta(N)) = \delta(\sigma(N)).$$

A less stringent relation will be obtained later.

(v) The multiplication function has many properties such as associativity which we might not wish to assume of our function H. In this case the multiplication function occurring in the definition of Gd will have to be replaced not by H but by some other function E which will be related to H in a way we describe later.

Making these replacements we obtain the following generalized definitions of Fr, Fd and Fu.

$$Fr(N) = [N = L \rightarrow B;;H(N, Fr(\delta_N))] \qquad (1)$$

$$Fd(N) = Gd(N, B)$$

where

$$Gd(N, A) = [N = L \rightarrow A;;Gd(\delta_N, E(N, A))] \qquad (2)$$

$$Fu(N) = Gu(N, L, B)$$

where

$$Gu(N, M, A) = [N = M \rightarrow A;;Gu(N, \sigma_M, H(\sigma_M, A))] \qquad (3)$$

where we remember that δ_N and σ_M are abbreviations for $\delta(N)$ and $\sigma(M)$.

What is the relation between the functions H and E? One can readily see from definitions (1) and (2) that

$$Fr(N) = H(N, H(\delta_N, H(\delta_N^2, \ldots, H(\delta^{\lambda-1}_N, B) \ldots)))$$

and

$$Fd(N) = E(\delta^{\lambda-1}_N, \ldots, E(\delta_N^2, E(\delta_N, E(N, B))) \ldots)$$

where δ_N^2 means $\delta(\delta(N))$, etc., and λ is the smallest integer such that $\delta_N = L$.

We wish to have $Fr(N) = Fd(N)$ for all relevant N. A sufficient condition on H and E for this to be true is that, for all α, β and γ,

$$H(\alpha, B) = E(\alpha, B)$$

and

$$H(\alpha, E(\beta, \gamma)) = E(\beta, H(\alpha, \gamma)) \qquad (4)$$

where it should be noted that B is a special constant of our universe, so the first of these equations only implies equality of the functions H and E if their second arguments take this special value.

An intuitive proof of the sufficiency of equations (4) is easy. By the first of equations (4) the inner E in the above expansion of $Fd(N)$ can be replaced by H. Repeated use of the second of equations (4) will move this H to the front. Now repeat this whole process on the new inner E, and so on until there are no E's left. We will then have Fr, thus "proving" that $Fr(N) = Fd(N)$ if equations (4) are satisfied.

We therefore state the following two theorems:

If equations (4) are satisfied then $Fr(N)$, defined by equation (1), is the same function as $Fd(N)$, defined by equations (2).

If equations (5) are satisfied then $Fr(N)$, as defined by equation (1), is the same function as $Fu(N)$, defined by equations (3).

$$\sigma(\delta(N)) = \delta(\sigma(N)) = N. \qquad (5)$$

These theorems should include statements about the values of N for which the functions Fr, Fd, and Fu are defined.

Further remarks on the generalization

Recursive definition (1) is equivalent to a recursive flow chart or ALGOL program, whilst recursive definitions (2) and (3) are equivalent to iterative programs; they are in the iterative form defined by McCarthy (1962) in Section 9. We therefore see that we can provide a mechanical procedure for converting a recursive definition which fits the form of equation (1) to an iterative form, so long as we can produce either the function E satisfying equations (4) or the function σ satisfying equations (5). Notice that equations (2) are essentially the scheme for definition by primitive recursion.

As another example which fits the form of equation (1) consider the interpretation in which N is a list, L and B are both the empty list and we have:

$\delta(N)$ is the list obtained by deleting the first member of list N.

$H(N, A)$ is the list obtained by adding the first member of the list N at the end of list A.

Then it is clear that $Fr(N)$ is the function that reverses the order of symbols on the list.

Can we produce the functions E and σ? Consider the function E defined by:

$E(N, A)$ is the list obtained by adding the first member of list N at the front of list A.

Then equations (4) are satisfied and we have produced an equivalent iterative definition (2).

The function σ satisfying equation (5) does not exist; to satisfy $\sigma(\delta(N)) = N$ the function σ must add to the list $\delta(N)$ the same term that $\delta(N)$ removed. However, if we add a second argument to σ, which will be the original list, then σ can perform that function.

We therefore modify equations (3) and (5) as follows:

$$Fu(N) = Gu(N, L, B)$$

where

$$Gu(N, M, A) = [N = M \rightarrow A;;Gu(N, \sigma(M, N), H(\sigma(M, N), A))] \qquad (3')$$

$$\sigma(\delta(N), N) = N \quad \text{and} \quad \delta(\sigma(M, N)) = M. \qquad (5')$$

It will still be true that $Fr = Fu$ if equation (5) is replaced by equation (5′).

We can now give the function σ for the list processing case. It is: $\sigma(M, N)$ is defined if and only if the list N is the list M preceded by one or more symbols, the value of $\sigma(M, N)$ is then the list obtained by adding to the front of the list M that member in front of M where it occurs as a sublist in N.

With this modification of the equations (3) and (5) the function σ always exists and can be defined by the recursive definition (in iterative form)

$$\sigma(M, N) = [\delta_N = M \rightarrow N;;\sigma(M, \delta_N)].$$

Example 3: approximate integration

A recursive definition may be used in a programming language that does not allow recursion by explicitly setting up a stack and saving all necessary intermediate results in this stack at appropriate points in the program, later restoring them, i.e. we include the mechanism necessary to deal with recursion as part of our program rather than using a mechanism provided for us in the system. We set up such an equivalence and later prove that the two definitions define the same function. We use an example which does not fit the schema of equation (1).

We wish to define the function $I(a, b)$, an approximation to the integral of some function $f(x)$ from a to b. We do not wish to vary the function f and so do not go to the trouble of including f as a third argument of the function I and of other functions to be introduced. We have some function $S(\alpha, \beta)$ which gives an approximation to the integral from α to β which may or may not be acceptable. We have a predicate $P(\alpha, \beta)$ which is true if $S(\alpha, \beta)$ is an acceptable approximation, and is false otherwise. $I(a, b)$ is then obtained by repeatedly dividing intervals into two equal halves until $P(\alpha, \beta)$ is satisfied in all intervals (α, β), and then adding up the contributions $S(\alpha, \beta)$ from all intervals.

A recursive definition of $I(a, b)$ is then

$$Ir(a, b) = [P(a, b) \rightarrow S(a, b);;Ir(a, a \oplus b) + Ir(a \oplus b, b)] \qquad (6)$$

where $a \oplus b$ is $(a + b)/2$.

If we have available the list processing functions of LISP, see McCarthy *et al.* (1962), then we can give an iterative definition that is equivalent to (6) in several ways. One such is

$$Ii(a, b) = f(0,\{a, b\})$$

where

$$f(V, L) = [\text{null}(\text{cdr}_\text{L}) \rightarrow V;\;\; P(\text{car}_\text{L}, \text{cadr}_\text{L}) \rightarrow f(V + S\,(\text{car}_\text{L}, \text{cadr}_\text{L}), \text{cdr}_\text{L});; f(V, ps(\text{car}_\text{L} \oplus \text{cadr}_\text{L}, \text{L}))] \tag{7}$$

$\{a, b\}$ is the list with two members a and b;

car_L and cadr_L are the first and second members of the list L;

$\text{null}(\text{cdr}_\text{L})$ is true if the list L has exactly one member; and we define

$$ps(V, L) = \text{cons}(\text{car}_\text{L}, \text{cons}(V, \text{cdr}_\text{L})) \tag{8}$$

i.e. $ps(V, L)$ is the list obtained by inserting V after the first member of L.

This recursive definition, in iterative form, was obtained by applying McCarthy's process to a flow chart of a program to obtain the approximate integral by an iterative process; the reader may easily deduce this flow chart for himself. The value of V is the value of the approximate integral from a to the first point on list L. The remaining members of L are points obtained by subdivision, i.e. are points up to which we must integrate if Ii is to be exactly the same function as Ir. The last member of list L is b itself.

We shall later prove the equivalence of Ii and Ir. In order to do this we need to modify their definitions and it is convenient to do this now. Remembering the intended meaning of V it is clear that we should be able to prove the equation $f(V, L) = V + f(0, L)$. If this equation is used throughout (7) the function f will always have its first argument zero, and we can therefore use a function of one variable instead of two.

We therefore define

$$Ij(a, b) = g(\{a, b\})$$

where

$$g(L) = [\text{null}(\text{cdr}_\text{L}) \rightarrow 0;\;\; P(\text{car}_\text{L}, \text{cadr}_\text{L}) \rightarrow S(\text{car}_\text{L}, \text{cadr}_\text{L}) + g(\text{cdr}_\text{L});; g(ps(\text{car}_\text{L} \oplus \text{cadr}_\text{L}, L))]. \tag{9}$$

This is another definition of our function, but of course not one in iterative form.

One difficulty in proving the equivalence of Ir and Ij is that Ij is defined in terms of a function g with a different domain from Ir. It is trivial to change the definition of Ir so that it has one argument $\{a, b\}$ instead of two, a and b, but it will still only be defined on lists with two members whereas g is defined on lists with any number of members (apart from the empty list). A good heuristic principle to use in this kind of situation is: try and extend the definition of Ir to the domain of g in such a way that the two functions become equivalent over this wider domain. Such an extension to the definition of Ir is

$$h(L) = [\text{null}(\text{cdr}_\text{L}) \rightarrow 0;; Ir(\text{car}_\text{L}, \text{cadr}_\text{L}) + h(\text{cdr}_\text{L})] \tag{10}$$

We shall later prove that $g(L) = h(L)$ and also that $Ii(a, b) = Ij(a, b)$; it is then trivial to prove that $Ir(a, b) = Ii(a, b)$.

Proofs

We now give proofs, using recursion induction, of the following:†

(A) if equations (4) are satisfied then $Fr = Fd$ (equations (1) and (2))

(B) if equations (5) are satisfied then $Fr = Fu$ (equations (1) and (3))

(C) $Ii = Ij$ (equations (7) and (9))

(D) $g = h$ (equations (9) and (10))

(E) $Ii = Ir$ (equations (6) and (7)).

The principle of recursion induction states that if we have an equation, or set of equations, which over a certain domain D is a legal recursive definition of some function, and if we have two previously defined functions which both satisfy these defining equations, then the two functions are equivalent over D. Loosely speaking a recursive definition must define a unique function.

As yet we have no techniques for proving that a recursive definition is legal over some domain D. It is intuitively obvious that the factorial function definitions are legal over the domain of non-negative integers, but do not define any values for negative integers. In this paper we shall ignore the admittedly important question of what is the domain of legality of our recursive definitions. However, in all cases this is obvious and it would be a not very difficult task to set up certain schema as being defined to be legal definitions over certain domains (essentially these are properties of the basic functions "predecessor" and "cdr") and certain rules giving transformations which preserve legality and thus prove the legality of all our recursive definitions.

Proof A. $Fr = Fd$

We first prove the following lemma:

$$Gd(\delta_N, H(N, A)) = H(N, Gd(\delta_N, A)). \tag{11}$$

The key to proving this seems to be to replace N on both sides (except where it occurs as an argument of the function δ) by a new variable S. One can then obtain a recursive equation in N satisfied by both sides of equation (11) in which S occurs as a parameter.

Define

$$\psi(N, S, A) = Gd(N, H(S, A)) \quad \text{and} \quad \chi(N, S, A) = H(S, Gd(N, A)).$$

Then using equation (2)

$$\begin{aligned} \psi(N, S, A) &= [N = L \rightarrow H(S, A);; Gd(\delta_N, E(N, H(S, A)))] \\ &= [N = L \rightarrow H(S, A);; Gd(\delta_N, H(S, E(N, A)))] \quad \text{(use equation (4))} \\ &= [N = L \rightarrow H(S, A);; \psi(\delta_N, S, E(N, A))] \end{aligned}$$

† By writing $f = g$ where f and g are function names we mean that the two functions are equivalent in the sense explained in the introduction to this paper.

and using equation (2) again

$$\chi(N, S, A) = [N = L \to H(S, A);;H(S, Gd(\delta_N, E(N, A)))]$$
$$= [N = L \to H(S, A);;\chi(\delta_N, S, E(N, A))].$$

We have thus shown that both ψ and χ satisfy the same recursive definition; therefore they are the same function. Substituting δ_N for N and N for S proves the lemma.

In equation (11) substitute B for A and then use the first of equations (4); this gives

$$Gd(\delta_N, E(N, B)) = H(N, Gd(\delta_N, B)).$$

Substitute B for A in equation (2) and use the above equation; this gives

$$Gd(N, B) = [N = L \to B;;H(N, Gd(\delta_N, B))]$$

or

$$Fd(N) = [N = L \to B;;H(N, Fd(\delta_N))].$$

This equation has the same form as equation (1) and so we have proved $Fr = Fd$ by recursion induction.

Note that throughout this proof we could have had a general predicate $P(N)$ in place of the predicate $N = L$; thus we could have replaced $N = L$ by $P(N)$ in equations (1) and (2).

Proof B. $Fr = Fu$

Define

$$Gr(N, M, A) = [N = M \to A;;H(N, Gr(\delta_N, M, A))] \quad (12)$$

so that

$$Fr_N = Gr(N, L, B).$$

We now prove that $Gr = Gu$. Note that although in the last proof we could have extended the definition of Gd to have a third argument, L, in exactly this manner, this would not have helped us for proof A. It would not have been true that $Gr = Gd$, only that they were equal for the special value B of their last argument. In this case, however, the extension does help; it is true that $Gr = Gu$.

Define

$$\chi(N, M, A) = [N = M \to A;;H(N, Gu(\delta_N, M, A))] \quad (13)$$

i.e. the right-hand side of the definition of Gr with Gr changed to Gu.

Then using equation (3)

$$\chi(N, M, A) = [N = M \to A;\delta_N = M \to H(N, A);; H(N, Gu(\delta_N, \sigma_M, H(\sigma_M, A)))]$$
$$= [N = M \to A;;\chi(N, \sigma_M, H(\sigma_M, A))]$$

where we have used equations (5) in the form $\delta_N = M \equiv N = \sigma_M$. Therefore by recursion induction $\chi = Gu$ (as this last is of the same form as equation (3) defining Gu).

Replace $\chi(N, M, A)$ by $Gu(N, M, A)$ in equation (12) and we then have Gu satisfying the defining equation of Gr, and so we have proved $Gr = Gu$ by recursion induction.

Substituting L for M and B for A gives $Fr = Fu$.

Proof C. $Ii = Ij$

The function Ij is defined recursively; the definition of Ii is an iterative definition counting down. Their equivalence may be proved directly by a similar proof to proof A or we may assume the result of proof A and deduce the equivalence of Ii and Ij. This latter may be done as follows.

Equations (7) and (9) are not directly in the form of equations (2) and (1); this is because the list L of equations (7) and (9) is not the list of points through which we have to move by using the function cdr; there may have to be interpolations. We can define the correct stepping function and therefore make the following definitions:

$$\delta(L) = [P(\text{car}_L, \text{cadr}_L) \to \text{cdr}_L;;\delta(pps_L)]$$

where

$$pps(L) = ps(\text{car}_L \oplus \text{cadr}_L, L) \quad \text{(see equation (8))}$$
$$g'(L) = [\text{null}(\text{cdr}_L) \to 0;;S(\text{car}_L, \text{car}(\delta_L)) + g'(\delta_L)]$$
$$f'(V, L) = [\text{null}(\text{cdr}_L) \to V;;f'(V + S(\text{car}_L, \text{car}(\delta_L)), \delta_L)].$$

Now using the definition of δ in the definition of g' we get

$$g'(L) = [\text{null}(\text{cdr}_L) \to 0;P(\text{car}_L, \text{cadr}_L) \to S(\text{car}_L, \text{cadr}_L) + g'(\text{cdr}_L);;S(\text{car}_L, \text{car}(\delta(pps_L))) + g'(\delta(pps_L))].$$

But using the fact that $\text{car}(pps_L) = \text{car}_L$ we see that the last expression of this conditional expression is $g'(pps_L)$. We have therefore proved by recursion induction that $g = g'$ and may prove in exactly the same way that $f = f'$.

If we remember the remark at the end of proof A, and in equations (1) and (2) specialize H and E to be given by

$$H(N, A) = E(N, A) = S(\text{car}_N, \text{car}(\delta_N)) + A$$

then from the equivalence of Fr and Fd we can deduce that

$$g'(L) = f'(0, L).$$

This, using $g = g'$ and $f = f'$, immediately leads to the equivalence of Ii and Ij.

Proof D. $g = h$

We can show directly that h (equation (10)) satisfies the recursive definition for g (equation (9)), i.e. that

$$h(L) = [\text{null}(\text{cdr}_L) \to 0;PP_L \to SS_L + h(\text{cdr}_L);; h(ps(\alpha_L, L)) \quad (14)$$

where

$$PP(L) = P(\text{car}_L, \text{cadr}_L)$$
$$SS(L) = S(\text{car}_L, \text{cadr}_L)$$

and

$$\alpha(L) = \text{car}_L \oplus \text{cadr}_L.$$

Now

$$\begin{aligned} h(ps(\alpha_L, L)) &= h(\text{cons}(\text{car}_L, \text{cons}(\alpha_L, \text{cdr}_L))) \\ &= Ir(\text{car}_L, \alpha_L) + h(\text{cons}(\alpha_L, \text{cdr}_L)) \\ &\qquad \text{(by equation (10))} \\ &= Ir(\text{car}_L, \alpha_L) + Ir(\alpha_L, \text{cadr}_L) + h(\text{cdr}_L) \\ &\qquad \text{(by equation (10) again).} \end{aligned}$$

Also using equation (6) in equation (10) we get

$$\begin{aligned} h(L) = [&\text{null}(\text{cdr}_L) \rightarrow 0; PP_L \rightarrow SS_L + h(\text{cdr}_L);; \\ &Ir(\text{car}_L, \alpha_L) + Ir(\alpha_L, \text{cadr}_L) + h(\text{cdr}_L)] \end{aligned}$$

and using the previous expansion of $h(ps(\alpha_L, L))$ we have equation (14); we have therefore proved by recursion induction that $g = h$.

Proof E. $Ii = Ir$

From equation (10)

$$h(\{a, b\}) = Ir(a, b) + h(\{b\}) = Ir(a, b).$$

Therefore

$$\begin{aligned} Ir(a, b) &= h(\{a, b\}) \\ &= g(\{a, b\}) && \text{by proof D} \\ &= Ij(a, b) && \text{by equation (9)} \\ &= Ii(a, b) && \text{by proof C.} \end{aligned}$$

A general method

We have now proved the equivalence of several functions, both as original proofs above and as lemmas for these proofs. The purpose in doing this was to obtain some kind of a feel for the way these proofs go, and hopefully to pinpoint some particularly useful proof methods. The majority of the proofs above were easy to obtain, any reasonable approach leading to a proof; but proof A (and proof C which we have shown to be effectively the same as proof A) was much more difficult to find. We now give a general method which can be used in proving the equivalence of functions, and which applies to all our equivalences except the main equivalences of proofs A and C.

This method is a generalization of the technique used in proof B to show the equivalence of Gr and Gu. Let us now change our notation slightly, allowing only functions of a single variable so that we regard Gr (equation (12)) as being a function of x where x is a 3-list $\{N, M, A\}$.

The form of the defining equations for Gu and Gr (equations (3) and (12)) is

$$\begin{aligned} Gu(x) &= Eu(Gu(u_x), x) \qquad (15) \\ Gr(x) &= Er(Gr(r_x), x) \end{aligned}$$

where Eu and Er are given conditional expressions and

$$\begin{aligned} u(\{N, M, A\}) &= \{N, \sigma_M, H(\sigma_M, A)\} \\ r(\{N, M, A\}) &= \{\delta_N, M, A\}. \end{aligned}$$

Proof B then goes as follows.

Define

$$X(x) = Er(Gu(r_x), x)$$

whence

$$X(x) = Er(Eu(Gu(u.r_x), r_x), x). \qquad (16)$$

We introduce the notation $u.r_x$ standing for $u(r(x))$ and later extend the notation to $f.g.h_x$ to stand for $f(g(h(x)))$, etc.

But it is clear that $u.r_x = r.u_x$; using this and properties of the actual conditional expressions Er and Eu equation (16) becomes

$$X(x) = Eu(Er(Gu(r.u_x), u_x), x)$$

whence

$$X(x) = Eu(X(u_x), x).$$

Therefore by recursion induction $X_x = Gu_x$; using this in the definition of X we again have by recursion induction that $Gu_x = Gr_x$.

This proof worked because we had a relation between the functions u and r, the relation $u.r = r.u$, and could show by manipulations of conditional expressions that $Er(Eu(\Phi, r_x), x) = Eu(Er(\Phi, u_x), x)$. We can generalize this result to cases where we are given other relations between the function u and r.

Theorem

Given:

(1) A domain D.

(2) Two partial functions $u(x)$ and $r(x)$ with arguments taken from D.

(3) Two recursive definitions defining total functions $Gu(x)$ and $Gr(x)$ over the domain D.

$$\left.\begin{aligned} Gu(x) &= Eu(Gu(u_x), x) \\ Gr(x) &= Er(Gr(r_x), x) \end{aligned}\right\} \qquad (17)$$

where Eu and Er are conditional expressions.

(4) An equation

$$f^1.f^2. \ldots f^n_x = g^1.g^2. \ldots g^m_x \qquad (18)$$

where each f^i and g^i is either the function r or the function u and both sequences are not the same sequence. The equation is to be satisfied for all x in D, this to mean that either both sides of the equation are defined and have the same value or both are undefined.

(5) A relation R which well-orders D and has the properties that

$$\begin{aligned} &u(x)Rx && \text{if } u(x) \text{ is defined} \\ &r(x)Rx && \text{if } r(x) \text{ is defined.} \end{aligned}$$

It then follows from the transitivity of R that

$$\delta(x)Rx \qquad \text{if } \delta(x) \text{ is defined}$$

where

$$\begin{aligned} \delta(x) &= f^1.f^2. \ldots f^n_x \qquad (19) \\ &(= g^1.g^2. \ldots g^m_x). \end{aligned}$$

(6) The following two recursive definitions of functions $Gf(x)$ and $Gg(x)$

$$Gf(x) = Ef^n(Ef^{n-1} \ldots (Ef^1(Gf(\delta_x), f^2.f^3. \ldots .f^n_x), \\ f^3. \ldots .f^n_x), \ldots f^n_x), x)$$

$$Gg(x) = Eg^m(Eg^{m-1} \ldots (Eg^1(Gg(\delta_x), g^2.g^3. \ldots . g^m_x), \\ g^3. \ldots g^m_x), \ldots g^m_x), x) \quad (20)$$

where Ef^i is either the conditional expression Er or the conditional expression Eu depending on whether f^i is r or u, and similarly for Eg^i.
Then

(I) The functions Gf and Gg are defined for all x in D.

(II) For all x in D $Gu(x) = Gr(x)$ if and only if for all x in D $Gf(x) = Gg(x)$.

We note that equations (20) define both Gf and Gg at a point x by means of conditional expressions involving the defined function evaluated at the same point $\delta(x)$. In all cases tried, trivial manipulations showed that these were the same conditional expression, thus proving $Gf = Gg$ and thence by the theorem $Gu = Gr$.

We will prove only a special case of the theorem, but the following proof technique is obviously valid in general. Consider then the special case when equations (18) and (19) are particularized to be

$$\delta(x) = r.u_x = u.u.r_x.$$

This situation is illustrated in **Fig. 1** where points indicate possible arguments for our functions, the function u is indicated by sloping lines downward and to the left, the function r by sloping lines downward and to the right. Choosing 1 to be any point x we always have such a configuration, assuming the functions u and r to be defined at the relevant points.

In this case equations (20) become

$$Gf(x) = Eu(Er(Gf(\delta_x), u_x), x)$$
$$Gg(x) = Er(Eu(Eu(Gg(\delta_x), u.r_x), r_x), x). \quad (21)$$

The proof of I depends on having techniques for proving that a given proposed recursive definition really does define a function over some given domain D. This paper has not gone into this question at all, and so we do not give a formal proof; in fact, all the actual functions defined in this paper are "obviously" good definitions, and only depend on such properties as if we start with any list and keep performing the cdr function eventually we must obtain the null list.

Let us now prove II. First assume that for all x in D $Gu(x) = Gr(x)$.

Then we have

$$\begin{aligned} Gu(x) &= Eu(Gu(u_x), x) \\ &= Eu(Gr(u_x), x) \\ &= Eu(Er(Gr(r.u_x), u_x), x) \\ &= Eu(Er(Gu(\delta_x), u_x), x) \end{aligned}$$

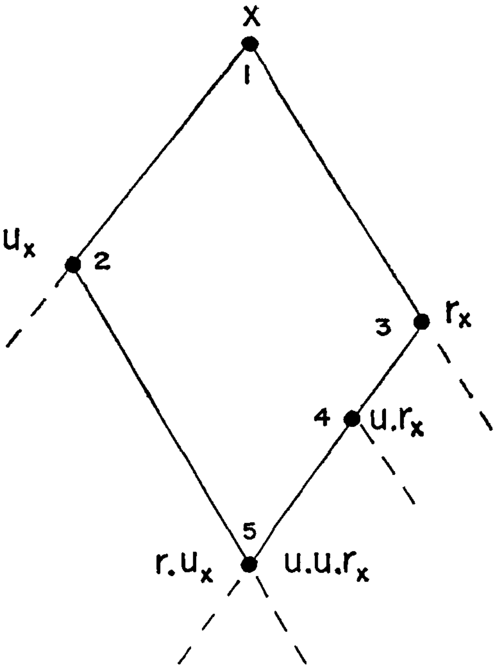

Fig. 1.—Configuration of points if $r.u_x = u.u.r_x$

and also

$$\begin{aligned} Gr(x) &= Er(Gr(r_x), x) \\ &= Er(Gu(r_x), x) \\ &= Er(Eu(Gu(u.r_x), r_x), x) \\ &= Er(Eu(Eu(Gu(u.u.r_x), u.r_x), r_x), x) \\ &= Er(Eu(Eu(Gr(\delta_x), u.r_x), r_x), x). \end{aligned}$$

We have shown that Gu satisfies the Gf equation and Gr satisfies the Gg equation. Therefore, by recursion induction $Gu = Gf$ and $Gr = Gg$ and it then follows that $Gf = Gg$.

Now assume that for all x in D $Gf(x) = Gg(x)$. We shall prove by the method of transfinite induction that for all x in D $Gr(x) = Gu(x)$. (The principle of transfinite induction states that if R is a well-ordering over a domain D and if $P(x)$ is a property such that assuming $P(z)$ is true for all z with zRx we can prove that $P(x)$ is true, then $P(x)$ must be true for all x in D.)

The property $P(x)$ which we shall prove true is that $Gu(x) = Gr(x) = Gf(x)$. Now take any x in D and we have

$$\begin{aligned} Gf(x) &= Eu(Er(Gf(r.u_x), u_x), x) && \text{by definition of } Gf \\ &= Eu(Er(Gr(r.u_x), u_x), x) && \text{by the induction hypothesis} \\ &= Eu(Gr(u_x), x) && \text{by definition of } Gr \\ &= Eu(Gu(u_x), x) && \text{by the induction hypothesis} \\ &= Gu(x) && \text{by definition of } Gu \end{aligned}$$

and in a similar manner

$$Gg(x) = Gr(x).$$

Whence $Gr(x) = Gu(x) = Gf(x)$, and our result follows by transfinite induction.

Examples of the use of the theorem

We were led to the theorem by considering the proof that $Gr = Gu$ where Gr and Gu are the specific functions defined by equations (12) and (3) (proof B). In this particular case x is a triplet $\{N, M, A\}$ and we have

$$u(x) = \{N, \sigma_M, H(\sigma_M, A)\} \quad \text{and} \quad r(x) = \{\delta_N, M, A\}$$

and obviously

$$u.r_x = r.u_x.$$

Corresponding to equations (20) we have the definitions

$$Gf(x) = \big[N = M \rightarrow A;;H(N, [\delta_N = M \rightarrow A;;Gf(\delta_x)])\big]$$

and

$$Gg(x) =$$

$$[N = M \rightarrow A;;[N = \sigma_M \rightarrow H(\sigma_M, A);;H(N, Gg(\delta_x))]\ \big]$$

Using equations (5) and simple properties of conditional expressions we see that these are really the same definition. Whence $Gf(x) = Gg(x)$ and by our theorem $Gu(x) = Gr(x)$.

The theorem, as we have stated it, cannot be used in proof D, the function g defined by equation (9) occurs twice in the conditional expression, each time with a different "decrementing" function. However, the theorem can be extended to this situation as follows.

In place of equations (17) assume our functions Gu and Gr are defined by equations of the form

$$Gu(x) = Eu(Gu(u_x^1), Gu(u_x^2) \ldots Gu(u_x^k), x)$$
$$Gr(x) = Er(Gr(r_x^1), Gr(r_x^2), \ldots Gr(r_x^m), x).$$

The first equation defines Gu at x in terms of Gu at all the points $\{u_x^1 \ldots u_x^k\}$. At each of these points we may choose to use further either the definition of Gu or the definition of Gr, or to do no further expansion. This will give a new enlarged set of points and we may continue this process. Similarly we could start with the definition of Gr. If we can find two such sequences of definitions of sets of points, both ending with the same final set of points, then we can define two new functions analogous to those of equations (20) which are equivalent functions if and only if $Gr = Gu$.

For example, take Gu to be g (equation (9)) and Gr to be h (equation (10)). Then $k = 2$, $u_x^1 = \text{cdr}_x$, $u_x^2 = ps(\text{car}_x \oplus \text{cadr}_x, x)$ and $m = 1$, $r_x^1 = \text{cdr}_x$. It can soon be proved that $r^1.r^1.u^2 = r^1$ and $u^1 = r^1$. Using these relations we can find a sequence of sets of points. Gu_x is defined in terms of $\{u_x^1, u_x^2\}$; now use the definition of Gr_x twice on the second point of this set giving $\{u_x^1, r^1.r^1.u_x^2\}$ or just $\{r_x^1\}$. This is precisely the point given by one application of the Gr definition, and so we have our desired two sequences. The corresponding two function definitions are:

$$Gf(x) = Eu(Gf(\delta_x), Er(Er(Gf(\delta_x), r^1.u_x^2), u_x^2), x)$$

and

$$Gg(x) = Er(Gg(\delta_x), x)$$

where $\delta(x)$ is r_x^1, i.e. $\text{cdr}(x)$ and

$$Eu(\phi, \psi, x) \text{ is } [\text{null}(\text{cdr}_x) \rightarrow 0;$$
$$P(\text{car}_x, \text{cadr}_x) \rightarrow S(\text{car}_x, \text{cadr}_x) + \phi;;\psi]$$

and

$$Er(\phi, x) \text{ is } [\text{null}(\text{cdr}_x) \rightarrow 0;;Ir(\text{car}_x, \text{cadr}_x) + \phi].$$

Some manipulations of the conditional expressions, and use of equation (8) shows that we really have Gf and Gg both satisfying the same recursive definition, and so as before we can use the theorem to show that g and h are the same function.

Many further examples of the use of the theorem could be given, but there are also many cases in which we cannot use the theorem because of the impossibility of finding the desired relation between the functions. If we define a function Gr by extending equation (1) thus:

$$Gr(N, A) = [N = L \rightarrow A;;H(N, Gr(\delta_N, A))]$$

we cannot use our theorem to try to prove $Gr = Gd$ (equation (2)). [Note that this is not even true is general, but stronger conditions on H and E than equations (4) may be imposed to make this true.]

We have the form of equations (15) (writing Gd instead of Gu) where x is $\{N, A\}$, u_x is $\{\delta_N, E(N, A)\}$ and r_x is $\{\delta_N, A\}$. It can soon be proved that there are not two different compositions of the functions u and r that are equal for all x.

References

McCarthy, J. (1960). "Recursive Functions of Symbolic Expressions and their Computation by Machine," *Comm. ACM*, Vol. 3, p. 184.

McCarthy, J. (1962). "Towards a Mathematical Science of Computation," IFIP Munich Conference, Amsterdam: North Holland Publishing Company, p. 21.

McCarthy, J. (1963). "A Basis for a Mathematical Theory of Computation," *Computer Programming and Formal Systems*, Ed. Braffort, P., and Hirschberg, D., Amsterdam: North Holland Publishing Company, p. 33.

McCarthy, J., *et al.* (1962). *Lisp 1.5 Programmer's Manual*, M.I.T. Press.