THE MC CARTHY'S RECURSION INDUCTION PRINCIPLE: « OLDY » BUT « GOODY »

L. KOTT (1)

ABSTRACT - We present here a very fruitful tool for proving properties of LISP functions. We implement the ancient, but quite natural and elegant, Recursion Induction Principle stated by J. Mc Carthy by using the famous fold/unfold method elaborated by R. Burstall and J. Darlington.

We thus obtain a very simple and flexible method for proving theorems about LISP functions; we call it the Mc Carthy method. Furthermore the method is machine oriented and we implement it in a conversational system. We do not make any comparison with the R. Boyer and J. Moore theorem - prover since our system is not automatic. But our system is implementable in a wide range of machines and we expect to implement our method in the R. Burstall and J. Darlington system. We then shall have a very powerful system which might perform program synthesis and proofs of program properties simultaneously (in a way parallel to that followed by Z. Manna and R. Waldinger).

In this paper we apply our method (by hand) to give the proof of two properties: associativity of the append operation between lists and idempotence of the reverse operation.

1. Introduction.

In 1962 J. Mc Carthy stated a recursion induction principle for proving properties of recursive programs [6]. We recall with an example the basic idea of this simple and, in some sense, naive principle (cf. [6]). Let us consider the recursive definition of addition over the positive integers

$$P(m, n) \Leftarrow \textit{if } m=0 \textit{ then } n \textit{ else } P(m-1, n)+1 \textit{ fi} \tag{1}$$

— Received: September 9, 1981.

(1) Université Paris VII LITP 2, place Jussieu F-75251 Paris Cedex 05 France.

and assume that we want to prove the property

$$\forall m \, \forall n \, P(m, n)+1 = P(m, n+1).$$

To prove this we introduce two new functions Q and R defined by

$$Q(m, n) \Leftarrow P(m, n)+1 \tag{2}$$

$$R(m, n) \Leftarrow P(m, n+1) \tag{3}$$

and we show that each function satisfies the following recursive definition

$$F(m, n) \Leftarrow \textit{if } m=0 \textit{ then } n+1 \textit{ else } F(m-1, n)+1 \textit{ fi}. \tag{4}$$

From this fact we deduce the equality of the functions Q and R (over the domain of data where they both terminate) and the property

$$\text{(P0)} \quad \forall m \, \forall n \, P(m, n)+1 = P(m, n+1).$$

This method is very attractive because, in spite of its name, it does not involve any induction principle. Unfortunately this method is not very easy to use because the search for the common recursive definition (like (4)) is somewhat difficult.

In order to solve this problem we propose a practical tool — called the Mc Carthy method — using the well-known fold/unfold method (initiated by R. Burstall and J. Darlington and Z. Manna and R. Waldinger [2, 8]) to try to find the common recursive definition. This tool is well fitted to applicative programming languages as LISP because it was developed in the framework of recursive programs.

Our aim is to provide to LISP programmers (beginners and experts) a tool for studying properties of their programs and increasing the knowledge about programs and how they actually work.

To contrast our system to Boyer and Moore's one we must give the following precisions:

— expressiveness is the same: both systems deal with recursive functions over lists or natural integers;

— Boyer and Moore's system is a powerful automatic system for proving properties about LISP programs. Our system is not automatic and its power is equal to the « power » of the user;

— last, but not least, our system may be implemented in a wide range of

computers, since its executable code rooms 23 K bytes. Furthermore it is already implemented in a HB 68 under MULTICS and LSI 11/23.

Before describing our method we must give more details about the programming language. Next we present an example of using the Mc Carty method, then a formal presentation and we finish with a second example.

2. Programming language.

We use a subset of pure LISP [7] which has as primitives nil, cons, car, cdr and cond. With these primitives we can define recursive functions as pointed out in [1]. Nevertheless, to avoid many brackets we shall use instead of cond the usual conditional *if ... then ... else ... fi* and the predicate nil; thus instead of cond (x, y, z) we shall write *if* nil (x) *then* z *else* y *fi*.

We express the programs in a recursive style as in [2]. We need the following:

X — a set of variable $x, y, z, x_1, \ldots$

PF — a set of primitives with zero or more arguments.

RF — a set of recursive function symbols noted $A, B, C, \ldots, F_1, F_2, \ldots$

Term — a term is built in the usual way out of primitives, variables, and recursive function symbols.

Recursive definition — a recursive definition is written $F(x_1, \ldots, x_n) \Leftarrow s$, where s is an expression, F a recursive function symbol with n arguments.

For example, to define the append operation between two lists we write the following recursive definition

(5) $\quad A(x, y) \Leftarrow$ *if* nil (x) *then* y *else* cons (car (x), A (cdr (x), y)) *fi*.

In some proofs we need some classical properties of primitives e. g. cdr (cons (x, y)) is equal to y (cf. [1]).

3. Informal description.

Let us consider the append function defined by (5) and assume we want to prove

$$\forall x \, \forall y \, \forall z \;\; A(\text{cons}(x, y), z) = \text{cons}(x, A(y, z)).$$

To prove this, we introduce two new recursive function symbols defined by

$$B(x, y, z) \Leftarrow A(\text{cons}(x, y), z) \tag{6}$$

$$C(x, y, z) \Leftarrow \text{cons}(x, A(y, z)). \tag{7}$$

By using the fold/unfold method we perform a transformation of (6) through the following definitions:

$$B(x, y, z) \Leftarrow \textit{if } \text{nil}(\text{cons}(x, y)) \textit{ then } z \textit{ else } \text{cons}(\text{car}(\text{cons}(x, y)), A(\text{cdr}(\text{cons}(x, y)), z)) \textit{ fi} \tag{8}$$

by unfolding the symbol A occurring in (6); we reach (9)

$$B(x, y, z) \Leftarrow \textit{if } \text{nil}(\text{cons}(x, y)) \textit{ then } z \textit{ else } \text{cons}(x, A(y, z)) \textit{ fi} \tag{9}$$

because car (cons (x, y)) is equal to x and cdr (cons (x, y)) to y.

But nil (cons (x, y)) is false so we obtain

$$B(x, y, z) \Leftarrow \text{cons}(x, A(y, z)). \tag{10}$$

Now if we consider definitions (7) and (10) we see that they are identical up to the recursive function symbols B and C. This fact has two consequences:

(i) the functions defined by (7) and (10) are equal

(ii) if we admit the transformation performed from (6) to (10) is correct, that is to say that the functions defined by (6) and (10) are equal, then the functions defined by (6) and (7) are equal.

Thus, we have proved the property

$$\text{(P1)} \quad \forall x \, \forall y \, \forall z \quad A(\text{cons}(x, y), z) = \text{cons}(x, A(y, z)).$$

Let us go further in the presentation of our method by showing

$$\forall x \, \forall y \, \forall z \quad A(x, A(y, z)) = A(A(x, y), z)$$

which expresses the associativity of append function. As above we define two new recursive functions

(11) $$D\,(x, y, z) \Leftarrow A\,(x, A\,(y, z))$$

(12) $$E\,(x, y, z) \Leftarrow A\,(A\,(x, y), z).$$

By unfolding the outermost occurrence of A in (11) we reach (13)

(13) $D\,(x, y, z) \Leftarrow$ *if* nil (x) *then* $A\,(y, z)$ *else* cons (card (x), A (cdr (x), $A\,(y, z))$) *fi*

then by folding

(14) $D\,(x, y, z) \Leftarrow$ *if* nil (x) *then* $A\,(y, z)$ *else* cons (car (x), D (*cdr* (x), y, z)) *fi*.

Now we deal with the function E. From (12) by unfolding the innermost occurrence of A we obtain

(15) $E\,(x, y, z) \Leftarrow A$ (*if* nil (x) *then* y *else* cons (car (x), A (cdr (x), y)) *fi*, z)

then by a usual law about *if ... then ... else ... fi* that we shall call distributivity over conditional

(16) $E\,(x, y, z) \Leftarrow$ *if* nil (x) *then* $A\,(y, z)$ *else* A (cons (car (x), A (cdr (x), y)),z) *fi*

but the property P1 enables us to write

(17) $E\,(x, y, z) \Leftarrow$ *if* nil (x) *then* $A\,(y, z)$ *else* cons (car (x), A (A (cdr (x), y), z)) *fi*

and, finally, by folding we reach

(18) $E\,(x, y, z) \Leftarrow$ *if* nil (x) *then* $A\,(y, z)$ *else* cons (car (x), E (cdr (x), y, z)) *fi*.

Again, if we compare definitions (14) and (18), we realize their identity up to the symbols D and E. So, if the transformations from (11) to (14) and from (12) to (18) are correct, we know that functions defined by (11) and (12) are equal and the property holds

(P2) $\forall x\, \forall y\, \forall z \quad A\,(x, A\,(y, z)) = A\,(A\,(x, y), z).$

4. Formal presentation.

We recall the rules for transforming recursive definitions defined by R. Burstall and J. Darlington. Given a set of recursive definitions we have the

following:

(i) *Definition.* Introduce a new recursive definition whose left-hand side is written $G(x_1, \ldots, x_n)$, where G is a new symbol. For example, (6), (7), (11) and (12) are definitions.

(ii) *Unfolding.* If $F_1(x_1, \ldots, x_m) \Leftarrow s_1$ and $F_2(x_1, \ldots, x_n) \Leftarrow s_2$ are recursive definitions and there is some occurrence in s_2 of an instance of $F_1(x_1, \ldots, x_m)$, replace it by the corresponding instance of s_1, obtaining s_3; then add the definition $F_2(x_1, \ldots, x_n) \Leftarrow s_3$. For example, unfolding with (5) takes (11) to (13).

(iii) *Folding.* If $F_1(x_1, \ldots, x_m) \Leftarrow s_1$ and $F_2(x_1, \ldots, x_n) \Leftarrow s_2$ are recursive definitions and there is some occurrence in s_2 of an instance of s_1, replace it by the corresponding instance of $F_1(x_1, \ldots, x_m)$, obtaining s_3; then add the definition $F_2(x_1, \ldots, x_n) \Leftarrow s_3$. For example, folding with (12) takes (17) to (18).

(iv) *Laws.* We may transform a recursive definition by using on its right-hand side any laws we have about primitive functions, obtaining a new equation. For example, laws about car, cdr and cons enable us to rewrite (8) as (9).

Starting with a system S of recursive definitions, we transform one or more recursive definitions with the above rules and we reach a system S' of new recursive definitions, for the recursive function symbols. A problem left open by R. Burstall and J. Darlington [2] was the correctness of the system that is to say the equality of functions defined by S and S'. We have studied this problem in the framework of algebraic semantics of recursive program schemes developed by M. Nivat [3, 10], and given a lot of sufficient conditions.

Here, we do not deal with program schemes but LISP programs. That means we have chosen an interpretation which interpretes the symbols of primitive functions (elements of PF) as LISP base operators with their usual properties. Under this assumption, as a corollary of the results mentioned above, the following proposition holds:

PROPOSITION 1. Assume we transform a system S of recursive definitions into a system S' by the fold/unfold method. This transformation is correct as soon as the number of unfolding is greater or equal than the number of folding.

For example, the proposition 1 ensures the transformations from (6) to (10) and from (11) to (14).

Let be S a set of recursive definitions, and t and s two terms. Assume we want to prove the equality

$\forall x_1, \ldots, \forall x_n \; t = s$ ($x_1, \ldots, x_n$ are the variables) (occurring in t and s).

We introduce two new recursive function symbols, say G and H, and two new systems of recursive definitions

$$S_1\begin{cases} G(x_1, \ldots, x_n) \Leftarrow s \\ S \end{cases} \qquad S_2\begin{cases} H(x_1, \ldots, x_n) \Leftarrow t \\ S \end{cases}$$

If we may transform S_1 and S_2 into S_1' and S_2'

$$S_1\begin{cases} G(x_1, \ldots, x_n) \Leftarrow s' \\ S \end{cases} \qquad S_2\begin{cases} H(x_1, \ldots, x_n) \Leftarrow t' \\ S \end{cases}$$

such that s' and t' are identical up to the symbols G and H (i. e. t' is s' in which G is replaced by H) and the transformations are correct, then we have proved the desired equality by the *Mc Carthy method.* For example, we proved the property (P1) by this method.

Now we define a new transformation rule:

(v) *Properties.* We may transform a recursive definition by using on its right-hand side any properties we have proved by the *Mc Carthy method* about the recursive functions, obtaining a new equation. For example, the property (P1) enables us to rewrite (16) as (17). We call this system of transformations the generalised fold/unfold method and we have studied it in [5] too. As for its correctness the proposition obtained from proposition 1 by replacing the words « fold/unfold method » by « generalised fold/unfold method », holds [5].

We shall keep the name Mc Carthy method when we use the generalised fold/unfold method. For example, we have proved the property (P2) by this new version of the Mc Carthy method because the transformation from (12) to (18) is correct.

5. Second example.

We define the reverse function by

(19) $$R(x) \Leftarrow \textit{if } \text{nil}(x) \textit{ then } \text{nil} \textit{ else } A(R(\text{cdr}(x)), \text{car}(x)) \textit{ fi}$$

and we are going to prove

$$\forall x \; R(R(x)) = x$$

(20) $$F(x) \Leftarrow R(R(x))$$

(21) $F(x) \Leftarrow R$ (*if* nil (x) *then* nil *else* A (R (cdr (x)), car (x)) *fi*)

by unfolding of the outermost occurrence of R in (20); then by law

(22) $F(x) \Leftarrow$ *if* nil (x) *then* R (nil) *else* R (A (R (cdr (x)), car (x))) *fi*.

We see that we need a property about R and A and, indeed, we are going to prove the property

$$\forall x \, \forall y \; R(A(x, y)) = A(R(y), R(x)).$$

This step looks very like the generalisation step of Boyer and Moore theorem-prover. Let us define

(23) $$G(x, y) \Leftarrow R(A(x, y))$$

(24) $$H(x, y) \Leftarrow A(R(y), R(x)).$$

From (23) by unfolding of the symbol A we obtain

(25) $G(x, y) \Leftarrow R$ (*if* nil (x) *then* y *else* cons (car (x), A (cdr (x), y)) *fi*).

By distributivity over conditional we reach

(26) $G(x, y) \Leftarrow$ *if* nil (x) *then* $R(y)$ *else* R (cons (car (x), A (cdr (x), y))) *fi*

by unfolding the rightmost occurrence of R we obtain

(27) $G(x, y) \Leftarrow$ *if* nil (x) *then* $R(y)$ *else if* nil (cons (car (x), A (cdr (x), y)))
then nil *else* A (R (cdr (cons (car (x), A (cdr (x), y)))),
car (cons (car (x), A (cdr (x), y)))) *fi fi*.

But for any lists x, y nil (cons (x, y)) is false, car (cons (x, y)) is equal to x and cdr (cons (x, y)) to y; thus we have

(28) $G(x, y) \Leftarrow$ *if* nil (x) *then* $R(y)$ *else* A (R (A (cdr (x), y)), car (car (x))) *fi*

(29) $G(x, y) \Leftarrow$ *if* nil (x) *then* $R(y)$ *else* A (G (cdr (x), y), car (x)) *fi*

with a folding. The transformation is correct since there are 2 unfolding and 1 folding. Now from (24) we reach by unfolding the rightmost occurrence of the symbol R

(30) $H\,(x, y) \Leftarrow A\,(R\,(y),\ \textit{if}\ \text{nil}\,(x)\ \textit{then}\ \text{nil}\ \textit{else}\ A\,(R\,(\text{cdr}\,(x)), \text{car}\,(x))\ \textit{fi})$

Again by distributivity over conditional we obtain

(31) $H\,(x, y) \Leftarrow \textit{if}\ \text{nil}\,(x)\ \textit{then}\ A\,(R\,(y), \text{nil})\ \textit{else}\ A\,(R\,(y), A\,(R\,(\textit{cdr}\,(x)), \text{car}\,(x)))\textit{fi}.$

By the property (P2) and the obvious property (P3)

$$\text{(P3)}\quad \forall x\ A\,(x, \text{nil}) = x$$

which is provable by the Mc Carthy method too, we obtain

(32) $H\,(x, y) \Leftarrow \textit{if}\ \text{nil}\,(x)\ \textit{then}\ R\,(y)\ \textit{else}\ A\,(A\,(R\,(y), R\,(\text{cdr}\,(x))), \text{car}\,(x)))\ \textit{fi}$

and, by folding,

(33) $H\,(x, y) \Leftarrow \textit{if}\ \text{nil}\,(x)\ \textit{then}\ R\,(y)\ \textit{else}\ A\,(H\,(\text{cdr}\,(x), y), \text{car}\,(x))\ \textit{fi}.$

The transformation from (24) to (33) is correct since there is 1 unfolding and 1 folding. Again definitions (29) and (33) are identical up to the symbols G and H and, since the performed transformations are correct, we have proved the property (P4)

$$\text{(P4)}\quad \forall x\ \forall y\ R\,(A\,(x, y)) = A\,(R\,(y), R\,(x)).$$

We use (P4) to transform (22) into (34)

(34) $F\,(x) \Leftarrow \textit{if}\ \text{nil}\,(x)\ \textit{then}\ R\,(\text{nil})\ \textit{else}\ A\,(R\,(\text{car}\,(x)), R\,(R\,(\text{cdr}\,(x))))\ \textit{fi}.$

To save some lines we admit that with 2 unfoldings and laws we may write

(35) $F\,(x) \Leftarrow \textit{if}\ \text{nil}\,(x)\ \textit{then}\ \text{nil}\ \textit{else}\ A\,(\text{car}\,(x), R\,(R\,(\text{cdr}\,(x))))\ \textit{fi}.$

By unfolding the symbol A we obtain

(36) $F(x) \Leftarrow$ *if* nil (x) *then* nil *else if* nil (car (x)) *then* $R(R(\text{cdr}(x)))$ *else* cons (car (car (x))), A (cdr (car (x))), $R(R(\text{cdr}(x)))$) *fi fi.*

But nil (car (x)) is false, car (car (x)) is equal to car (x) and cdr (car (x)) to nil, so

(37) $F(x) \Leftarrow$ *if* nil (x) *then* nil *else* cons (car (x)), A (nil, $R(R(\text{cdr}(x)))$) *fi*

and by another unfolding of A we reach

(38) $F(x) \Leftarrow$ *if* nil (x) *then* nil *else* cons (car (x)), $R(R(cdr(x)))$ *fi*

and by folding

(39) $F(x) \Leftarrow$ *if* nil (x) *then* nil *else* cons (car (x)), F (cdr (x))) *fi.*

The transformation from (20) to (39) is correct since there are 5 unfoldings and 1 folding is correct. Furthermore if we admit that the identity function over the domain of lists may be defined by

(40) $I(x) \Leftarrow$ *if* nil (x) *then* nil *else* cons (car (x)), I (cdr (x))) *fi.*

Then we notice that (39) and (40) are identical up to the symbols F and I and we have established the property (P5)

(P5) $\forall x \; R(R(x)) = I(x) = x.$

To conclude we hope the reader is convinced that this method is useful and that there is a simple and flexible system which uses it. Furthermore if we want a more automatic system it is possible to adapt the Burstall-Darlington system to perform the Mc Carthy method. By the way we could use the same system for developing recursive programs and proving properties about them, which, as pointed out by Z. Manna and R. Waldinger [9], seems a very fruitful approach.

BIBLIOGRAPHY

[1] R. BOYER, J. MOORE: *Proving theorems about LISP function.* J. of ACM (22) **1** (1975), 129-144.

[2] R. BURSTALL, J. DARLINGTON: *A transformation system for developing recursive programs,* J. of ACM (24) **1** (1977), 44-67.

[3] B. COURCELLE, M. NIVAT: *Algebraic families of interpretations,* 17th FOCS, Houston, (1976).

[4] L. KOTT: *About transformation system: a theoretical study, in « Transformations de Programmes »,* (1978), 232-247.

[5] L. KOTT: *Des substitutions dans les systèmes d'équations algébriques sur le magma,* Doctoral Dissertation, Univ. Paris VII, (1979).

[6] J. MAC CARTHY: *A basis for a mathematical theory of computation,* in « Computer Programming and Formal Systems », (1963).

[7] J. MC CARTHY et al.: *LISP* 1.5 *Programmer's Manual,* M. I. T. Press Cambridge, Mass., (1962).

[8] Z. MANNA, R. WALDINGER: *Knowledge and reasoning in program synthesis,* Artif. Intel. J. (6) **2** (1975), 175-208.

[9] Z. MANNA, R. WALDINGER: *A deductive approach to program synthesis,* ACM ToPLaS (2) **1** (1980), 90-121.

[10] M. NIVAT: *Interprétation universelle d'un schéma de programme récursif,* Informatica (7) **1** (1977), 9-16.