MECHANICAL CONSTRUCTION OF A NEW EFFICIENT FLATTEN.

Yves Kodratoff,+ Jean-Pierre Jouannaud++

+ G.R. 22 du C.N.R.S., Institut de Programmation 4 Place Jussieu 75230 PARIS CEDEX 05.

++ Université de Nancy 2, 42 Avenue de la Libération 54000 NANCY.

1. MCFLAT.

An efficient function FLATTEN is "well-known" and due to Mc CARTHY[5]. We give below a version of it, using the predicates NULL and ATOM.

```
(DE MCFLAT(X)(FOO X NIL))
(DE FOO(X Z)(COND
              ((NULL X)Z)
              ((ATOM X)(CONS X Z))
              (T(FOO(CAR X)(FOO(CDR X)Z)))))
```

In order to study the complexity of this function we shall count the number of times FOO is called when one evaluates (MCFLAT X) where X is a list containing n atoms and m couples of parenthesis. In an intuitive way, consider first the list (A) which leads to 3 calls of FOO (the initial one and then 2 recursive calls). When an atom is added to it , say X becomes (A B), 5 calls to FOO are necessary, i.e. we add 2 new calls to FOO per new atom. The same is true when one considers ((A)), i.e. we add 2 calls by new nesting level. This means that FOO is called 2n+2m-1 times by MCFLAT.

2. MQFLAT : transforming programs contructed from examples.

2.1.-

Our methodology for getting programs from input-output examples is described in[1, 2, 3] . In principle, it consists into 4 steps.

The input sequence is transformed into a predicate sequence, the output sequence is tranformed into a computation traces sequence, each of the new sequences is transformed to sets of recursion relations which can finally be proved[2, 6]to be equivalent to a program.

We shall study here the example of the top-level COPY function. The input-output sequence is $\{x_o=(A)\rightarrow f(x_o)=(A);\ x_1=(A\ B)\rightarrow f(x_1)=(A\ B);x_2=(A\ B\ C)\rightarrow f(x_2)=(A\ B\ C);...\}$.

We deduce from it the following predicates and computation traces sequences:

$\{p_o(x)=(ATOM(CDR\ X))\rightarrow f_o(X)=(CONS(CAR\ X)NIL);$

$p_1(x)=(ATOM(CDDR\ X))\rightarrow f_1(X)=(CONS(CAR\ X)(CONS(CADR\ X)NIL));$

$p_2(X)=(ATOM(CDDDR\ X))\rightarrow f_2(X)=(CONS(CAR\ X)(CONS(CADR\ X)(CONS(CADDR\ X)NIL)));\ ...\}$.

Matching p_i and p_{i+1} leads trivially to the recursion relation $p_{i+1}(X)=p_i(CDR(X))$.

Matching f_i and the "tail" of f_{i+1} (as the arrows show) leads to the recursion relation: $f_{i+1}(X)=CONS(CAR(X), f_i(CDR(X)))$ which we shall rather write for further transformation purposes : $f_{i+1}(X) = h(X, f_i(CDR(X)))$ where $h(X, Z) = (CONS(CAR\ X)Z)$.

We finally obtain a program whose stopping conditions are given by the first predicate and the first trace and whose recursive body is given by the recursion relations:

```
(DE COPY(X)(COND
          ((ATOM(CDR X))(CONS(CAR X)NIL))
          (T(H X(COPY(CDR X))))))
(DE H(X Z)(CONS(CAR X)Z))
```

2.2.

The programs we thus obtain are generally defined for a specific input domain which is here the flat lists(they have only atoms at their top-level). As a matter of fact, it can be proved that they are defined for any list but they act at the top-level only. We have described in [4a] how to obtain from this program F an other program F' which is recusively defined on all nesting levels of the list. We describe in [4b] how to obtain directly (FLATTEN(F' X)), i.e. a program which executes the action of F at all nesting levels and flattens its result.

We shall not detail here these transformations : they are quite cumbersome to describe in the general case. From the intuitive point of view, we can simply say that the form of the synthesized programs allowsthe reasoning : F' is like F except that it tests if (CAR X) (or other $(CAD^kR\ X)$ expressions) in the stopping branch are atomic or not. If yes then F' is identical to F, if not then F' calls itself again, applied to the corresponding non-atomic expression.

Applied to COPY, this transformation leads to MQFLAT:

```
(DE MQFLAT(X)(FQQ X NIL))
(DE FQQ(X Z)(COND                                          FQQ1
          ((ATOM(CDR X))(COND
                    ((ATOM(CAR X))(CONS(CAR X)Z))          FQQ2
                    (T(FQQ(CAR X)Z))))
          (T(HQQ X(FQQ(CDR X)Z)))))                        FQQ3
(DE HQQ(X Z)(COND
          ((ATOM(CAR X))(CONS(CAR X)Z))                    FQQ4
          (T(FQQ(CAR X)Z))))
```

When (MQFLAT '(A)) is evaluated, only 1 call to FQQ1 is needed. When one adds one atom in the list then (MQFLAT '(A B)) calls FQQ1 then HQQ (thus FQQ4), i.e. each new atom induces 2 new recursive calls. When one adds a nesting level, MQFLAT applied to ((A)) needs a call to FQQ1 and a call to FQQ2 ,i.e. each nesting level adds only 1 supplementary recursive call. It follows that if x contains n atoms and m pairs of parenthesis, 2n+m-2 calls to FQQ and HQQ are needed.

2.3. An other(inefficient) FLATTEN.

From the above example , one might believe that we implicitely claim that our methodology leads always to good programs. This would be wrong and we feel it interesting enough that automatically constructed programs are not automatically the worse possible!

One can get an other COPY from the examples by matching "root to root" f_i and f_{i+1}.

The recurrence relations are slightly more complicated since one needs to generalize $f_i(X)$ to $g_i(X, NIL)$ and look for recursion relations between $g_i(X,Z)$ and $g_{i+1}(X,Z)$ where $g_i(X,Z)$ is the same expression as $f_i(X)$ except that the NIL of $f_i(X)$ is replaced by Z.

Once this generalization is made, one finds quite easily the recursion relations :

$f_i(X)=g_i(X,NIL)$;

$g_o(X,Z)=(CONS(CAR\ X)Z)$, $g_{i+1}(X,Z)=g_i(X,h_i(X,Z)$;

$h_o(X,Z)=(CONS(CADR\ X)Z)$, $h_{i+1}(X,Z)=h_i(CDR(X),Z)$.

One also remarks that the X of g_i recurs as $X \to X$ while the X of p_i (the domain predicates) recurs as $X \to (CDR\ X)$ so that we need another variable (we shall call it XX) in order to express the domain recurrence relations.

Finally, we obtain the program :

```
(DE COPY2(X)(G2 X X NIL))
(DE G2(X XX Z)(COND
                ((ATOM(CDR XX))(CONS(CAR X)Z))
                (T(G2 X(CDR XX)(H2 X XX Z)))))
(DE H2(X XX Z)(COND
                ((ATOM(CDDR XX))(CONS(CADR X)Z))
                (T(H2(CDR X)(CDR XX)Z))))
```

We apply the same transformation to this new COPY and find :

```
(DE MKFLAT(X)(FKK X X NIL))
(DE FKK(X XX Z)(COND
                ((ATOM(CDR XX))(COND
                                ((ATOM(CAR X))(CONS(CAR X)Z))
                                (T(FKK(CAR X)(CAR X)Z))))
                (T(FKK X(CDR XX)(HKK X XX Z)))))
(DE HKK(X XX Z)(COND
                ((ATOM(CDDR XX))(COND
                                ((ATOM(CADR X))(CONS(CADR X)Z))
                                (T(FKK(CADR X)(CADR X)Z))))
                (T(HKK(CDR X)(CDR XX)Z))))
```

MKFLAT is clearly of degree 2 polynomial complexity, it runs quite slowly as compared with the other two.

One should notice that the good performance of MQFLAT originates from the fact that the H function which embedds COPY is a constant function while the H2 function embedded in G2 is not constant. This is a very special case and one should in the general case choose the "root to root" matching rather than the matching of section 2.1..

3. Discussion.

It is difficult to say that our program is "better" than Mc CARTHY's since the stopping conditions are not the same and since we use cross-recursion. We therefore do not claim that each LISP system should implement MQFLAT. It is nevertheless surprising that a transformation which is systematic enough to be implemented(and shall be soon implemented) , i.e. a program that can be automatically synthesized can well challenge the programs due to very skillful programmers.

This illustrates a point which a matter of deep contests among the practisers of Artificial Intelligence : does A.I. rely or not on simulation of behaviour ?

It is clear that we never analysed our own way of programming in order to discover MQFLAT . On the contrary , this program comes from considerations on necessities internal to program synthesis by machine. In the AISB society we must admit that we have a tendency to accentuate the AI rather than the SB.

Aknowledgment : We were induced to this work through discussions with J S. MOORE.

REFERENCES.

[1] J.-P. JOUANNAUD, Y. KODRATOFF " Characterization of a class of functions synthesized from examples by a SUMMERS like method using a "BMW" matching technique" Proc. IJCAI-79 Tokyo(1979) and Pub. Univ. Nancy C.R.I.N. (1979).

[2] Y. KODRATOFF " A class of functions synthesized from a finite number of examples and a LISP program scheme", Interntl. J. of Comp. and Information Sciences Vol 8, n°6, (1 9 7 9).

[3] Y. KODRATOFF, J. FARGUES "A sane algorithm for the synthesis of LISP functions from examples : the BOYER and MOORE algorithm", Proc. AISB meeting , Hamburg (1978), pp 169-175.

[4] a - Y. KODRATOFF, J.-P. JOUANNAUD " Construction automatique à partir d'exemples de fonctions de listes récursivement définies pour tous les niveaux d'imbrication des listes" Actes congrès AFCET/IRIA Reconnaissance des formes et Intelligence Artificielle Toulouse (1979).

b - Y. KODRATOFF, J.-P. JOUANNAUD "The transformation of top-level defined to all nesting levels defined list functions" Pub. Univ. Nancy, C.R.I.N. (1979).

[5] cited by J S. MOORE " A tour through a working theorem prover", 4th Workshop on Artificial Intelligence Bad Honnef (1979).

[6]P.D. SUMMERS " A methodology for LISP program construction from examples" J. ACM (1977), 24, 161-175.