ГОДИШНИК НА СОФИЙСКИЯ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ"

ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
Книга 1 — Математика
Том 83, 1989

ANNUAIRE DE L'UNIVERSITE DE SOFIA „ST. KLIMENT OHRIDSKI"

FACULTE DE MATHEMATIQUES ET INFORMATIQUE
Livre 1 — Mathématiques
Tome 83, 1989

SCHEMES FOR RECURSION ELIMINATION IN BACKUS' FP-SYSTEMS

Atanas Radensky, Milena Djambazova

Атанас Раденски, Милена Джамбазова. СХЕМЫ ДЛЯ УСТРАНЕНИЯ РЕКУРСИИ В БЭКУСОВСКИХ FP-СИСТЕМАХ

В статье рассмотрены трансформации с применением метода табюлации рекурсивных функций в среде FP-систем Бекуса. Корректность трансформации доказана методом индукций неподвижной точки. Рассмотрены следующие классы рекурсивных функций: рекурсия относительно одного целого аргумента; рекурсия относительно целого элемента списка; полная рекурсия; взаимно-рекурсивные функции.

Atanas Radensky, Milena Djambazova. SCHEMES FOR RECURSION ELIMINATION IN BACKUS' FP-SYSTEMS

In this paper, the idea of tabulation is explored in order to develop transformations of recursive functions in Backus' FP-sytems into iterative ones. Transformation schemes are described in the paper and the correctness of some of them is proved by means of fixpoint induction. The following classes of recursion are considered: recursion with respect to one integer argument, recursion with respect to an integer element of a list, multiple recursion, mutual recursion.

1. INTRODUCTION

A well known approach to recursion elimination consists in the development of equivalent schemes which allow transformation of recursive programs into iterative ones [11, 5, 8]. Transformation of linear recursive schemes (schemes of De Backer and Scott) into iterative ones in purely syntactical studied [12,13]. Programs which support automated transformation of recursion into iteration are considered [4]. On the other hand, some authors develop methods which are usable "manually", by human programmers [1]. Recursion elimination is considered on an implementation level as well in [9]. Frequently, the low efficiency of evaluation of a function is due to the fact that the function is applied many times to the same argument during

the evaluation. A well known example in this respect is the Fibonacci function:

```
function Fib(n : integer) : integer;
begin
  if (n = 0) or (n = 1) then Fib := 1
  else Fib := Fib(n-1) + Fib(n-2);
end;
```

The application of *Fib* to *n* causes one application of *Fib* to $n-1$, two different applications of *Fib* to $n-2$, and so on. Suppose, any value of the function *Fib* is memorized in a special table after its first computation. This value can be used without recomputation any time the function should be applied to the same argument. In addition, the order of computation can be changed from "top down" to "bottom up". In such a way the recursive function *Fib* can be replaced by an iterative one:

```
function Fib(n:integer) : integer;
var table : array [0..max] of integer;
    k : integer;
begin table[0] := 1; table[1] := 1;
      for k := 2 to n do
        table[k] := table[k-1] + table[k-2];
      Fib := table[n];
end;
```

Obviously, the most general variant of tabulation when all computed values are memorized may cause great memory overhead and is too inefficient. The more realistic case is when only some of the values are memorized for further use. It is clear that $table[k]$ depends only on $table[k-1]$ and $table[k-2]$, i.e. only on the two preceding values. Using this dependence, the above function can be easily transformed into more efficient one, in which the table is of size 2. (Since the last function mentioned above is well known, it is not presented here.)

This example illustrates techniques for recursion elimination known as tabulation [3]. This technique consists in transforming recursive programs into iterative ones by means of additional tables. In this paper, the idea of tabulation is explored in order to develop transformations of recursive functions in Backus' FP-sytems [2] into iterative ones. As it will be shown, the iterative variant of the function itself implements some kind of tabulation. Actually, the function computes some values in some order, memorizes them during some steps of the computation and uses the memorized values in the further computations. A number of schemes for recursion elimination in FP-systems have been already developed. Kieburtz and Shultis [6] formulate in the language of FP-systems and prove the validity of many schemes well-known from outside the FP-systems. Reddy and Jayaraman [10] develop an approach to recursion elimination based on the representation of recursive FP-functions in the form of infinite conditions.

In this paper some transformaion schemes are described and the correctness of some of them is proved by means of fixpoint induction. All transformations considered reduce recursive functions to tail-recursive ones. The transformation of tail-recursion to iteration is straightforward [6]. The following classes of recursion are considered:

1) Recursion with respect to one integer argument;
2) Recursion with respect to an integer element of a list;
3) Multiple recursion;
4) Mutual recursion.

2. REMARKS ON THE PROOF METHOD AND ON THE NOTATION

Suppose, f and g are recursively defined functions: $Def\ f = F(f)$ and $Def\ g = G(g)$. In the paper, some equivalence relations in the form $L(f) = R(g)$ are considered. Throughout the paper such equality is proved by the following variant of fixpoint induction (Scot's μ-induction) [7]:

Base. $L(\perp) = R(\perp)$.

Let $f_0 \subseteq f_1 \subseteq f_2 \subseteq \dots$ and $g_0 \subseteq g_1 \subseteq g_2 \subseteq \dots$ are two monotonicaly increasing sequence of functions, such that $L(f_i) = R(g_i)$. Then $L(\bigcup_{i=0}^{\infty} f_i) = R(\bigcup_{i=0}^{\infty} g_i)$.

Induction. It is demonstrated that the assumption $L(f) = R(g)$ implies $L(F(f)) = R(G(g))$.

Second step of the base is always true in FP-systems because all function and functionals are continuous [6,7,14,15].

As far as notations are concerned, it is supposed that the reader is familiar with those ones suggested by Backus in [2].

Throughout the paper, E denotes an arbitrary function, which has integer values when is defined. In addition, q, q_0, q_1, $\dots$, q_{n-1} denote arbitrary integers.

For the reader who is unfamiliar with the notation of FP-systems, suffice it to say that $\circ$ signifies the ordinary (left-associative) composition of functions; the construction $[f_1, f_2, \dots, f_n]$ is n-tuple of of functions f_i, $(1 \leqq i \leqq n)$. $\perp$ is everywhere undefined function, $\overline{x}$ signify constant function.

Some of the laws of FP-algebra we use are:

1. $\mathrm{id} \circ f = f \circ \mathrm{id} = f$
2. $f \circ (p \longrightarrow g; h) = p \longrightarrow f \circ g; f \circ h$
3. $(p \longrightarrow g; h) \circ f = p \circ f \longrightarrow g \circ f; h \circ f$
4. $k \circ [f_1, \dots, f_n] = f_k \quad (1 \leqq k \leqq n)$
5. $\perp \circ f = \perp$

3. RECURSION WITH RESPECT TO ONE INTEGER ARGUMENT

Let us consider functions whose definitions have the following form:

(R_1) $$Def\ f = \mathrm{eq}_0 \longrightarrow \overline{q};\ E \circ [f \circ \mathrm{sub}_1, \mathrm{id}];$$

Example. The definition of the *factorial* function can be given in the following form:

$$Def\ factorial = \mathrm{eq}_0 \longrightarrow \overline{1}; x \circ [factorial \circ \mathrm{sub}_1, \mathrm{id}];$$

Suppose, f is defined according to the scheme (R_1). If $f(n-1)$ is known, then the computation of $f(n)$ is straightforward. $f(n)$ can be computed starting from $f(0) = q$ and computing $f(i+1)$ from $f(i)$ for $i = 0, 1, \dots, n-1$. A simple table in the form $(f(i), i, n)$ can be used. (In the beginning, the table is $(q, 0, n)$.) For $i = 1, 2, \dots, n-1$ the following is executed:

— $f(i+1)$ is computed by means of $f(i)$, which is in the table;

— $f(i+1)$ replaces $f(i)$ in the table;

— the second component i of the table (the "counter") is increased by 1;

— it is checked whether the counter i is already equal to the third component n.

This approach allows the function f to be replaced by a function f', defined according to the following scheme:

$$(I_1)\qquad \begin{aligned} &Def\ f' = g \circ [\overline{q}, \overline{0}, \mathrm{id}];\\ &Def\ g = \mathrm{eq} \circ [2, 3] \longrightarrow 1;\ g \circ [E \circ [1, \mathrm{add}_1 \circ 2], \mathrm{add}_1 \circ 2, 3]; \end{aligned}$$

Example. In the case of the *factorial* function the above definitions look like this:

$$\begin{aligned} &Def\ factorial' = g \circ [\overline{1}, \overline{0}, \mathrm{id}];\\ &Def\ g = \mathrm{eq} \circ [2,3] \longrightarrow 1;\ g \circ [x \circ [1, \mathrm{add}_1 \circ 2], \mathrm{add}_1 \circ 2, 3]; \end{aligned}$$

The correctness of the transformation under consideration is established by the following theorem:

Theorem 1. *Let f be defined according to the scheme* (R_1) *and let f' be defined according to the scheme* (I_1). *Then $f = f'$.*

Proof. The equality $g \circ [q, 0, \mathrm{id}] = f$ is to be proved.

Base. $\bot \circ [\overline{q}, \overline{0}, \mathrm{id}] = \bot$.

Induction. Suppose $f = f'$. Substitute f' for f in the right side of the definition of f. It is necessary to prove that the resulting expression is equivalent to the right side of the definition of f':

$$(1)\qquad \mathrm{eq}_0 \longrightarrow \overline{q}; E \circ [f' \circ \mathrm{sub}_1, \mathrm{id}]$$

(Substitute the right side of the definition of f' for f'.)

$$= \mathrm{eq}_0 \longrightarrow \overline{q};\ E \circ [g \circ [\overline{q}, \overline{0}, \mathrm{id}] \circ \mathrm{sub}_1, \mathrm{id}]$$

$$= \mathrm{eq} \circ [2,3] \circ [\overline{q}, \overline{0}, \mathrm{id}] \longrightarrow 1 \circ [\overline{q}, \overline{0}, \mathrm{id}]; E \circ [g \circ [\overline{q}, \mathrm{sub}_1 \circ 1, \mathrm{id}] \circ \mathrm{sub}_1, \mathrm{id}]$$

In order to reduce the above expression, the following Lemma is used:

Lemma 1. *For arbitrary integers a, b:*

$$E \circ [g \circ [\overline{a}, \mathrm{sub}_1 \circ b, \mathrm{id}] \circ \mathrm{sub}_1, \mathrm{id}] = g \circ [E \circ [\overline{a}, \overline{b}], \overline{b}, \mathrm{id}].$$

According to the above lemma, expression (1) becomes:

$$\begin{aligned} &\mathrm{eq} \circ [2,3] \circ [\overline{q}, \overline{0}, \mathrm{id}] \longrightarrow 1 \circ [\overline{q}, \overline{0}, \mathrm{id}];\ g \circ [E \circ [\overline{q}, \overline{1}], \overline{1}, \mathrm{id}]\\ &= (eq \circ [2,3] \longrightarrow 1;\ g \circ [E \circ [1, \mathrm{add}_1 \circ 2], \mathrm{add}_1 \circ 2, 3]) \circ [\overline{q}, \overline{0}, \mathrm{id}]\\ &= g \circ [\overline{q}, \overline{0}, \mathrm{id}], \end{aligned}$$

and Theorem 1 is proved.

As far as lemma 1 is concerned, it can be proved by fixpoint induction, too:

Base. $E \circ [\bot \circ [\overline{a}, \mathrm{sub}_1 \circ \overline{b}, \mathrm{id}] \circ \mathrm{sub}_1, \mathrm{id}] = \bot \circ [E \circ [\overline{a}, \overline{b}], \overline{b}, \mathrm{id}]$.

Induction. Let denote $a' = E \circ [\overline{a}, \overline{b}]$, $b' = \mathrm{add}_1 \circ \overline{b}$. If a' and b' are integer constants, then according to the inductive hypothesis:

$$g \circ [E[\overline{a'}, \overline{b'}], \overline{b'}, \mathrm{id}] = E \circ [g \circ [\overline{a'}, \mathrm{sub}_1 \circ \overline{b'}, \mathrm{id}] \circ \mathrm{sub}_1, \mathrm{id}].$$

The same equality holds when $a' = \bot$ or $b' = \bot$, or the both.

Now, the right side of the equality under consideration can be unfolded and reduced as follows:

$$g \circ [E \circ [\overline{a}, \overline{b}], \overline{b}, \mathrm{id}] =$$

(definition of g)

$$= \mathrm{eq} \circ [\overline{b}, \mathrm{id}] \longrightarrow E \circ [\overline{a}, \overline{b}]; g \circ [E \circ [E \circ [\overline{a}, \overline{b}], \mathrm{add}_1 \circ \overline{b}], \mathrm{add}_1 \circ \overline{b}, \mathrm{id}]$$

$$= \mathrm{eq} \circ [\overline{b}, \mathrm{id}] \longrightarrow E \circ [\overline{a}, \mathrm{id}]; g \circ [E \circ [\overline{a'}, \overline{b'}], \overline{b'}, \mathrm{id}]$$

(inductive hypothesis)

$$= \mathrm{eq} \circ [\overline{b}, \mathrm{id}] \longrightarrow E \circ [\overline{a}, \mathrm{id}]; E \circ [g \circ [\overline{a'}, \mathrm{sub}_1 \circ \overline{b'}, id] \circ \mathrm{sub}_1, \mathrm{id}]$$

$$= \mathrm{eq} \circ [\overline{b}, \mathrm{id}] \longrightarrow E \circ [\overline{a}, \mathrm{id}]; E \circ [g \circ [E \circ [\overline{a}, \overline{b}], \overline{b}, \mathrm{id}] \circ \mathrm{sub}_1, \mathrm{id}]$$

(FP-algebra)

$$= E \circ [(\mathrm{eq} \circ [\overline{b}, \mathrm{id}] \longrightarrow \overline{a}; g \circ [E \circ [\overline{a}, \overline{b}], \overline{b}, \mathrm{id}] \circ \mathrm{sub}_1), \mathrm{id}]$$

$$= E \circ [(\mathrm{eq} \circ [\overline{b-1}, \mathrm{id}] \longrightarrow \overline{a}; g \circ [E \circ [\overline{a}, \overline{b}], \overline{b}, \mathrm{id}]) \circ \mathrm{sub}_1, \mathrm{id}]$$

(definition of g, folding)

$$= E \circ [g \circ [\overline{a}, \mathrm{sub}_1 \circ \overline{b}, \mathrm{id}] \circ \mathrm{sub}_1, \mathrm{id}].$$

The last expression is just the left side of the equality. This is the end of the proof of lemma 1.

Now, the following more general class of definitions is considered:

$$(\mathrm{R}_2) \qquad \begin{aligned} Def\ f = \mathrm{eq}_0 \longrightarrow \overline{q_0}; \mathrm{eq}_1 \longrightarrow \overline{q_1}; \ldots; \mathrm{eq} \circ [\overline{k-1}, \mathrm{id}] \longrightarrow \overline{q_{k-1}}; \\ E \circ [[f \circ \mathrm{sub}_1, f \circ \mathrm{sub}_1^2, \ldots, f \circ \mathrm{sub}_1^k], \mathrm{id}]. \end{aligned}$$

Example. The Fibonacci function is defined by a particular instance of the scheme (R_2), with $k = 2$, $E = + \circ 1$:

$$Def\ Fib = \mathrm{eq}_0 \longrightarrow \overline{1}; \mathrm{eq}_1 \longrightarrow \overline{1}; + \circ [Fib \circ \mathrm{sub}_1, Fib \circ \mathrm{sub}_1^2].$$

In the scheme (R_2), $f(n)$ depends on $f(n-1), f(n-2), \ldots, f(n-k)$. In that, the values of f for the first k natural numbers are known. It is convenient to use a table, which has the form

$$\langle\langle f(i), f(i-1), \ldots, f(i-k+1)\rangle, i, n\rangle.$$

In the beginning of the computation, the table has the following contents:

$$\langle\langle q_{k-1}, q_{k-2}, \ldots, q_0\rangle, k-1, n\rangle.$$

In order to compute $f(n)$, the values $f(k), f(k+1), f(k+2), \ldots$ are computed one after the other. In that, $f(i+1)$ is computed simply by application of E to the first element of the table, i.e. the list $\langle f(i), f(i-1), \ldots, f(i-k+1)\rangle$ of k preceding The value so obtained is appended to the left of the same list, and its rightmost component $f(i-k+1)$ is dropped. Finally, on each step of the computation the second component of the table (the counter) is increased by one. The computation stops when the counter becomes equal to n.

Using such kind of tabulation, f can be transformed into a tail-recursive function f', which is defined according to the following scheme:

$$(I_2')\quad \begin{aligned} &Def\ f = \mathrm{eq}_0 \longrightarrow \overline{q_0};\ \mathrm{eq}_1 \longrightarrow \overline{q_1};\ \ldots;\ \mathrm{eq} \circ [\overline{k-1}, \mathrm{id}] \longrightarrow \overline{q_{k-1}};\\ &\qquad\qquad g \circ [[\overline{q_{k-1}}, \overline{q_{k-2}}, \ldots, \overline{q_0}], \overline{k-1}, id],\\ &Def\ g = \mathrm{eq} \circ [2,3] \longrightarrow 1 \circ 1;\ g \circ [[E \circ [1, \mathrm{add}_1 \circ 2], 1 \circ 1, \ldots, (k-1) \circ 1],\\ &\qquad\qquad \mathrm{add}_1 \circ 2, 3]. \end{aligned}$$

Example. The Fibonacci function can be defined by a tail-recursive definition, following the scheme (I_2'):

$$\begin{aligned} &Def\ Fib' = \mathrm{eq}_0 \longrightarrow \overline{1};\ \mathrm{eq}_1 \longrightarrow \overline{1};\ g \circ [[\overline{1}, \overline{1}], \overline{1}, id],\\ &Def\ g = \mathrm{eq} \circ [2,3] \longrightarrow 1 \circ 1;\ g \circ [[+ \circ 1, 1 \circ 1], \mathrm{add}_1 \circ 2, 3]. \end{aligned}$$

Theorem 2. *Let f be defined according to the scheme (R_2) and let f' be defined according to the scheme (I_2'). Then $f = f'$.*

4. RECURSION WITH RESPECT TO AN INTEGER ELEMENT OF A LIST

There are many functions which are applied to lists in the form

$$\langle n, a_2, a_3, \ldots, a_r \rangle,$$

and which are defined recursively with respect to the first element n of the list (n is integer). Such functions can be defined according to the following scheme:

$$(R_3)\qquad Def\ f = \mathrm{eq}_0 \circ 1 \longrightarrow \overline{q};\ E \circ [f \circ [\mathrm{sub}_1 \circ 1, 2, \ldots, r], j].$$

Example. According to the above scheme, multiplication of natural numbers can be expressed by addition as follows:

$$Def\ mult = \mathrm{eq}_0 \circ 1 \longrightarrow \overline{0};\ + \circ [mult \circ [\mathrm{sub}_1 \circ 1, 2], 2].$$

Functions, defined according to the scheme (R_3), can be transformed to tail-recursive ones by means of a table in the form:

$$\langle f(i), i, \langle n, a_2, a_3, \ldots, a_r \rangle \rangle.$$

(When the computation starts, the content of the table is $\langle q, 0, \langle n, a_2, a_3, \ldots, a_r \rangle \rangle$.) The second component i of the table is the counter — the computation starts with a counter 0 and stops when the counter is equal to n.

A tail-recursive function f' which is equivalent to f can be defined according to the following scheme:

$$(I_3)\qquad \begin{aligned} &Def\ f' = g \circ [\overline{q}, \overline{0}, \mathrm{id}],\\ &Def\ g = \mathrm{eq} \circ [2, 1 \circ 3] \longrightarrow 1;\\ &\qquad g \circ [E \circ [1, j \circ [\mathrm{add}_1 \circ 2, 2 \circ 3, 3 \circ 3, \ldots, r \circ 3]], \mathrm{add}_1 \circ 2, 3]. \end{aligned}$$

Example. The scheme (I_3) determines the following tail-recursive definition of the multiplication function:

$$\begin{aligned} &Def\ mult' = g \circ [\overline{0}, \overline{0}, \mathrm{id}],\\ &Def\ g = \mathrm{eq} \circ [2, 1 \circ 3] \longrightarrow 1;\\ &\qquad g \circ [+ \circ [1, 2 \circ 3], \mathrm{add}_1 \circ 2, 3]. \end{aligned}$$

Theorem 3. *Let f be defined according to the scheme* (R_3) *and let f' be defined according to the scheme* (I_3). *Then $f \doteq f'$.*

Proof. *Base.* $\bot \circ [\overline{q}, \overline{0}, \mathrm{id}] = \bot$.

Induction. The substitution of f' for f in the right side of the definition of f leads to the following:

$$\begin{aligned}
&\mathrm{eq}_0 \circ 1 \longrightarrow \overline{q};\\
&\qquad E \circ [g \circ [\overline{q}, \overline{0}, \mathrm{id}] \circ [\mathrm{sub}_1 \circ 1, 2 \circ 3, 3 \circ 3, \ldots, r \circ 3], j]\\
&\qquad \text{(Here Lemma 2 — see below — is applied:)}\\
&= (\mathrm{eq} \circ [2, 1 \circ 3] \longrightarrow 1;\\
&\qquad\qquad g \circ [E \circ [1, j \circ [\mathrm{add}_1 \circ 2, 2 \circ 3, 3 \circ 3, \ldots, r \circ 3]],\\
&\qquad\qquad\quad \mathrm{add}_1 \circ 2, 3]) \circ [\overline{q}, \overline{0}, \mathrm{id}]\\
&= g \circ [\overline{q}, \overline{0}, \mathrm{id}],
\end{aligned}$$

and this is the end of the proof of Theorem 3.

Lemma 2. *For any integers q and a the following equality holds:*

$$\begin{aligned}
&E \circ [g \circ [\overline{q}, \overline{a}, \mathrm{id}] \circ [\mathrm{sub}_1 \circ 1, 2 \circ 3, 3 \circ 3, \ldots, r \circ 3], j]\\
&\quad = g \circ [E \circ [\overline{q}, j \circ [\mathrm{add}_1 \circ \overline{a}, 2 \circ 3, 3 \circ 3, \ldots, r \circ 3]], \mathrm{add}_1 \circ \overline{a}, \mathrm{id}].
\end{aligned}$$

The proof again can be carried out by fixpoint induction.

Example. A function, which computes the power when applied to a list $\langle n, a \rangle$ is defined according to the scheme (R_3):

$$Def\ power = \mathrm{eq}_0 \circ 1 \longrightarrow \overline{1}; * \circ [power \circ [\mathrm{sub}_1 \circ 1, 2], 2].$$

Its tail-recursive variant, obtained according the scheme (I_3) looks like this:

$$Def\ power' = g \circ [\overline{1}, \overline{0}, \mathrm{id}],$$

$$Def\ g = \mathrm{eq} \circ [2, 1 \circ 3] \longrightarrow 1;\ g \circ [* \circ [1, 2 \circ 3], \mathrm{add}_1 \circ 2, 3].$$

We introduce two slightly modified versions of the schemes (R_3) and (I_3):

(R_3') $$Def\ f = \mathrm{eq}_0 \circ 1 \longrightarrow \overline{q};\ E \circ [f \circ \mathrm{apndl} \circ [\mathrm{sub}_1 \circ 1, \mathrm{tl}], j],$$

$$Def\ f' = g \circ [\overline{q}, \overline{0}, \mathrm{id}],$$

(I_3') $$Def\ g = \mathrm{eq} \circ [2, 1 \circ 3] \longrightarrow 1;$$

$$g \circ [E \circ [1, j \circ \mathrm{apndl} \circ [\mathrm{add}_1 \circ 2, \mathrm{tl} \circ 3]], \mathrm{add}_1 \circ 2, 3].$$

The last two schemes are useful, for instance, when computing sums and products of the following kind:

$$p(i, a_1, a_2, \ldots, a_r) = sigma(n, a_1, a_2, \ldots, a_r),$$

$$p(i, a_1, a_2, \ldots, a_r) = pi(n, a_1, a_2, \ldots, a_r).$$

Corresponding recursive functions are defined according to the scheme (R_3') as follows:

$$\begin{aligned}
&Def\ sigma = eq0 \circ 1 \longrightarrow \overline{0};\\
&\qquad\qquad + \circ [p, sigma \circ \mathrm{apndl} \circ [\mathrm{sub}_1 \circ 1, \mathrm{tl}]],
\end{aligned}$$

$$Def\ pi = eq_0 \circ 1 \longrightarrow \overline{1};$$
$$* \circ [p, pi \circ apndl \circ [sub_1 \circ 1, tl]].$$

A tail-recursive version of *sigma* is easily obtained according to the scheme (I'_3):

$$Def\ sigma' = g \circ [\overline{0}, \overline{0}, id],$$
$$Def\ g = eq \circ [2, 1 \circ 3] \longrightarrow 1;$$
$$g \circ [+ \circ [1, p \circ apndl \circ [add_1 \circ 2, tl \circ 3]], add_1 \circ 2, 3].$$

A tail-recursive version of the function *pi* can be obtained in an analogous way.

5. FULL TABULATION

Let us consider a function f of an integer argument, such that:
— its value $f(0) = q$ is known;
— the value of $f(n)$ depends on n and $f(n-1), f(n-2), \ldots, f(0)$.
A function g can be considered, which tabulates all values of f:

$$g(n) = \langle f(n), f(n-1), \ldots, f(0) \rangle.$$

Hence, f can be defined by means of g as follows:

$$Def\ f = eq_0 \longrightarrow \overline{q};\ E \circ apndl \circ [id, g \circ sub_1],$$
$$Def\ g = eq_0 \longrightarrow [\overline{q}];\ apndl \circ [f, g \circ sub_1].$$

Since $f = 1 \circ g$, the computation of f is reduced to the computation of g. From the other hand, g can easily be replaced by a tail-recursive function:

$$Def\ g = g'[[\overline{q}], \overline{0}, id],$$
$$Def\ g' = eq \circ [2, 3] \longrightarrow 1;$$
$$g' \circ [apndl \circ [E \circ apndl \circ [add_1 \circ 2, 1], 1], add_1 \circ 2, 3].$$

The function g tabulates all values of f. Such kind of full tabulation is inefficient. Actually, it is not needed in practice. The idea behind it however can be used in order to implement some restricted forms of tabulation.

Let us consider again a function, whose value $f(n)$ depends only on n and $f(n-1), f(n-2), \ldots, f(n-k)$, as in scheme ($R_2$). Now we consider a slightly different scheme:

(R_4)
$$Def\ f = eq_0 \longrightarrow \overline{q_0};\ eq_1 \longrightarrow \overline{q_1}; \ldots\ eq \circ [\overline{k-1}, id] \longrightarrow \overline{q_{k-1}};$$
$$E \circ [id, f \circ sub_1, f \circ sub_1^2, \ldots, f \circ sub_1^k].$$

An equivalent tail-recursive definition follows:

(I_4)
$$Def\ f' = eq_0 \longrightarrow \overline{q_0};\ eq_1 \longrightarrow \overline{q_1}; \ldots;$$
$$eq \circ [\overline{k-1}, id] \longrightarrow \overline{q_{k-1}};\ 1 \circ g;$$
$$Def\ g = eq_0 \longrightarrow [\overline{q_0}];\ leq \circ [id, \overline{k-1}] \longrightarrow apndl \circ [f', g \circ sub_1];$$
$$tlr \circ apndl \circ [f', g \circ sub_1].$$

It can be demonstrated that the functions f and f', defined by the schemes (R_4) and (I_4), are equivalent.

Let us note that the function g can be easily defined a tail-recursive variant:

$$Def\ g = g' \circ [[\overline{q_{k-1}}, \overline{q_{k-2}}, \ldots, \overline{q_0}], \overline{k-1}, \mathrm{id}];$$
$$Def\ g' = \mathrm{eq} \circ [2,3] \longrightarrow 1;$$
$$g' \circ [\mathrm{apndl} \circ [E \circ [\mathrm{apndl} \circ [\mathrm{add}_1 \circ 2, 1]], \mathrm{tlr} \circ 1], \mathrm{add}_1 \circ 2, 3];$$

Examples. Let us consider again the factorial function:

$$Def\ factorial = \mathrm{eq}_0 \overline{1};\ x \circ [\mathrm{id}, factorial \circ \mathrm{sub}_1].$$

(The above definition slightly differs from the one considered earlier.) According to the above schemes, the same function can be defined with tail-recursion:

$$Def\ factorial' = \mathrm{eq}_0 \longrightarrow \overline{1};\ 1 \circ g;$$
$$Def\ g = g' \circ [[\overline{1}], \overline{0}, \mathrm{id}];$$
$$Def\ g' = \mathrm{eq} \circ [2,3] \longrightarrow 1;$$
$$g' \circ [\mathrm{apndl} \circ [x \circ [\mathrm{apndl} \circ [\mathrm{add}_1 \circ 2, 1]], \mathrm{tlr} \circ 1], \mathrm{add}_1 \circ 2, 3];$$

The same scheme can be applied to the Fibonacci function.

6. MUTUALLY RECURSIVE FUNCTIONS

Two mutually recursive functions f and g, defined as follows:

(R_5)
$$Def\ f = \mathrm{eq}_0 \longrightarrow \overline{q_1};\ E_1 \circ [f \circ \mathrm{sub}_1, g \circ \mathrm{sub}_1, \mathrm{id}];$$
$$Def\ g = \mathrm{eq}_0 \longrightarrow \overline{q_2};\ E_2 \circ [f \circ \mathrm{sub}_1, g \circ \mathrm{sub}_1, \mathrm{id}];$$

can be reduced to functions f' and g' according to the scheme below:

$$Def\ f' = h \circ [\overline{1}, \mathrm{id}]; \qquad Def\ g' = h \circ [\overline{2}, \mathrm{id}];$$
$$Def\ h = \mathrm{and} \circ [\mathrm{eq}_0 \circ 2, \mathrm{eq}_1 \circ 1] \longrightarrow \overline{q_1};\ \mathrm{and} \circ [\mathrm{eq}_0 \circ 2, \mathrm{eq}_2 \circ 1] \longrightarrow \overline{q_2};$$
(R_5')
$$t \circ [1, h \circ a, h \circ b, \mathrm{id}];$$
$$Def\ t = \mathrm{eq}_1 \circ 1 \longrightarrow E_1 \circ [2,3,4];\ E_2 \circ [2,3,4];$$
$$Def\ a = [\overline{1}, \mathrm{sub}_1]; \qquad Def\ b = [\overline{2}, \mathrm{sub}_1];$$

The function h can be reduced to a tail-recursive one. To do so, tables in the form

$$\langle\langle h(1,i), h(2,i)\rangle, \langle 1, n\rangle, i\rangle, \qquad \langle\langle h(1,i), h(2,i)\rangle, \langle 2, n\rangle, i\rangle,$$

are used. (The first table is used when f is to be computed, and the second — respectively when g is to be computed.) The tables allow to obtain an iterative variant g' of the above defined function g, and consequently — iterative variants

of f' and g':

$$Def\ f' = h' \circ [\overline{1}, \mathrm{id}]; \qquad Def\ g' = h' \circ [\overline{2}, \mathrm{id}];$$

$$Def\ h' = h'' \circ [[\overline{q_1}, \overline{q_2}], \mathrm{id}, \overline{0}];$$

$$Def\ h'' = \mathrm{and} \circ [\mathrm{eq} \circ [2 \circ 2, 3], \mathrm{eq}_1 \circ 1 \circ 2] \longrightarrow 1 \circ 1;$$

$$\mathrm{and} \circ [\mathrm{eq} \circ [2 \circ 2, 3], \mathrm{eq}_2 \circ 1 \circ 2] \longrightarrow 2 \circ 1;$$

$$h'' \circ [[E_1 \circ [1 \circ 1, 2 \circ 1, \mathrm{add}_1 \circ 3],$$

$$E_2 \circ [1 \circ 1, 2 \circ 1, \mathrm{add}_1 \circ 3]], 2, \mathrm{add}_1 \circ 3];$$

7. CONCLUSION

By means of tabulation, some more complicated recursive definitions can also be reduced to tail-recursive ones. Among them are, for instance, definitions in which recursion is with respect to several parts of the argument $\langle n, k \rangle$:

$$Def\ f = \mathrm{eq}_0 \circ 1 \longrightarrow \overline{q_1};\ \mathrm{eq}_0 \circ 2 \longrightarrow \overline{q_2};$$

$$E \circ [f \circ [\mathrm{sub}_1 \circ 1, 2], f \circ [1, \mathrm{sub}_1 \circ 2], \mathrm{id}].$$

Thus, tabulation techniques allow to obtain many interesting schemes for recursion removal in FP-systems.

REFERENCES

1. Arsac J., Y. Codratoff. Some techniques for recursion removal from recursive functions. *ACM Transactions on Programming Languages and Systems*, 4, 1982, 295–322.
2. Backus, J. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Communications of the ACM*, 21, 1978, 613–641.
3. Backus, J. W. The algebra of functional programs: functional level reasoning, linear equations and extended definitions. *In*: Formalization of programming concepts, ed. Diaz and Ramos, Springer-Verlag (1981).
4. Bird, R. Tabulation techniques for recursion removal. *Computing Surveys*, 12, 1980, 403–417.
5. Bustall, R., J. Darlington. A transformation system which automatically improves programs. *Acta Informatica*, 6, 1976, 41–60.
6. Hikita, T. On a class of recursive procedures and equivalent iterative ones. *Acta Informatica*, 12, 1979, 305–320.
7. Kieburtz, R., J. Shultis. Transformation of FP schemes. *In*: Proceedings of the conference on functional programming languages and computer architecture, Portmounth, New Hamphire, New York: ACM 1981 41–48.
8. Manna, Z. Mathematical theory of computation. New York, McGraw-Hill 1974.
9. Partsch, H., P. Pepper. A family of rules for recursion removal. *Information Processing Letters*, 5, 1976, 174–177.
10. Radensky, A. Computer, programming language, compiler. One nonconventional programming language , and its implementation. Sofia, Nauka i Izkustvo 1987 (in Bulgarian).
11. Reddy, U., B. Jayaraman. Theory of linear equations applied to program transformation. *In*: Proceedings of the eighth international joint conference on artificial intelligence, Karlsruhe (A. Bundy, ed.), Los Altos, California: Wil, 1, 10–16.
12. Trachtenbrot, M. Transformations of recursive programs as a mean of efficient program construction. *In*: Program optimization and transformation. Proceedings of a national seminar (A. Ershov, ed.), Novosibirsk, VC SO AN SSSR 1987, part 2, 119–136.

13. Williams, J. H. On the development of the algebra of functional program. *ACM Trans. on Prog. Lang. and Systems*, 4, 1982, 4, 733–757.
14. Патерсон М. С., С. Е. Хьюит. Сравнительная схематология. *В* кн.: Кибернетический сборник. Вып. 13(новая серия). М., „Мир", 1976, 62–72.
15. Гарланд С., Д. Лакхэм. Стандартные схемы, рекурсивные схемы и формальные языки. *В* кн.: Кибернетический сборник. Вып. 13(новая серия), М., „Мир", 73–119.

Received 13.03.1990