

E.R. 50

INSTITUT DE PROGRAMMATION

UP 55-65

place Jussieu, 75230 Paris Cedex 05

Tél. 333 25.25 ou 329 12.21 - Poste : 53 90

KODRATOFF Yves. Groupe "Structures de l'information" du C.N.R.S.

Length of the intended talk : 45 mn.

title : A methodology for recursion to iteration program transformation using a generalization technique.

In collaboration with J. ARSAC, we have devised a new methodology for recursion to iteration program transformation [1]. This methodology makes use of a generalization technique which is quite different from WEGBREIT's [6]. We shall describe our and WEGBREIT's methodologies for recursion to iteration program transformation with an emphasis on the powerfulness and the defects of the two methodologies. This recalls strongly also the celebrated results of BURSTALL and DARLINGTON [2] and we shall show that our methodology avoids the "eureka" they need.

1. BURSTALL and DARLINGTON.

We shall not describe their results since they are well-known. Let us simply recall an example (from DARLINGTON [3]).

Given the definition $f(x) \Leftarrow \text{if } x = 0 \text{ then } 0 \text{ else } f(x - 1) + x$, they can transform f to an iterative form by the use of their unfolding-folding technique provided a function $g(x,y) = f(x - 1) + y$ is given to their system. The knowledge of this function is an "eureka" which can be avoided with the two following methodologies.

2. WEGBREIT.

2.1. Example

The unique example given by WEGBREIT [6] is relative to the FIBONACCI function but we shall work out the preceding example :

$$f(x) \Leftarrow \text{if } x = 0 \text{ then } 0 \text{ else } \underbrace{f(x - 1) + x}_{(1)}$$

We know that the only x_1 such that $(x_1 = 1) = \text{True}$ has the value 1, that the inverse of the function $- 1$ is $+ 1$, so that we can uniquely compute

$$x_2 = x_1 + 1 = 2.$$

We can unfold $f(x)$ in the "context condition" $x \neq 1$ which leads to

$$f(x) \Leftarrow \text{if } x = 0 \text{ then } 0 \text{ else if } x = 1 \text{ then } f(1-1) + 1$$

$$\qquad \qquad \qquad \text{else } (f((x-1)-1) + x - 1) + x$$

Using the distributivity of + and - and the definition of $f : f(1) = 1$, we obtain :

$$f(x) \Leftarrow \text{if } x = 0 \text{ then } 0 \text{ else if } x = 1 \text{ then } 1 \text{ else } \underbrace{f(x-2) + 2x - 1}_{(2)}$$

We attempt a match of the expressions

(1) and (2) which leads to the contradictory substitutions:

$x \rightarrow x - 1$, $x \rightarrow 2x - 1$ and therefore fails. We accordingly generalize expression

(1) to (1') = $f(x-1) + y$. The unfolding of $f(x)$ applied to (1') leads to :

if $x = 1$ then y else $f(x-2) + y + x - 1 = (2')$.

The match of (1') and 2') succeeds because $x \rightarrow x - 1$ and $y \rightarrow y + x - 1$ are not contradictory.

Replacing $f(x-1) + y$ by the more general expression $g(x,y)$ we obtain :

$$f'(x) \Leftarrow \text{if } x = 0 \text{ then } 0 \text{ else } g(x,x)$$

$$g(x,y) \Leftarrow \text{if } x = 1 \text{ then } y \text{ else } g(x-1, y + x - 1)$$

$f'(x)$ and $f(x)$ gives same result but $f'(x)$ is "tail-recursive", i.e. readily transformable to an iteration by an application of the reverse of Mc CARTHY [5] transformation. The "eureka" has been transformed into a generalization induced by the failure of the match of (1) and (2)

2.2. Restrictions.

Let us study the more general scheme :

$f(x) \Leftarrow \text{if } a(x) \text{ then } b(x) \text{ else } h(f(e(x)), c(x))$. It shows that we need :

a.1. x_1 , such that $a(x_1) = \text{true}$ is unique and computable with the only knowledge of $a(x)$.

Examples.

- If $a(x) = \text{even}(x)$, x_1 is not unique

- If f is a function in two variables x and y , then $a(x,y) = (x=y)$ is not computable with the only knowledge of $a(x,y)$.

a.2. $e(x)$ is such that $e^{-1}(x)$ is defined and $e^{-1}(x_1)$ is unique. Example :
If $a(x) = (\text{CDR } x)$, $e^{-1}(x)$ is not defined.

a.3. $h(x)$ is not an if... then... else... function. If it were the case, then we would have a scheme of the type :

$$f(x) \Leftarrow \text{if } a(x) \text{ then } b'(x) \text{ else if } a'(x) \text{ then } h'(f(e'(x)), c'(x))$$

$$\qquad \qquad \qquad \text{else } h''(f(e(x)), c(x))$$

and the unfolding of $f(x)$ in $h''(f(e(x)), c(x))$ would lead to expressions containing h' the match of which with h'' is possible for very particular

functions.

a.4. $h(h(x,y),z) = h(x,h(y,z))$: this is the classical associative property for these transformations.

a.5. When $f(x)$ is not everywhere defined, one introduces the value ω , "undefined", in order to express the fact that if $f(z) = \omega$ the computation of $f(z)$ needs an infinite computation loop. In order to prove that the generalized scheme is not less defined than $f(x)$ one needs : $h(\omega,x) = \omega$ for all x .

3. ARSAC and KODRATOFF.

3.1 Example.

Once more, let us start from

$$f(x) \Leftarrow \text{if } x=0 \text{ then } 0 \text{ else } f(x-1)+x.$$

We notice that $x+0=x$ for all x (in the following we shall say that $+$ has a "neutral element") so that we can write

$$f(x)+0 = f(x) \Leftarrow \text{if } x=0 \text{ then } 0 \text{ else } f(x-1)+x.$$

WE DO NOT UNFOLD $f(x)$ but attempt a match of the new expression of f with its recursive definition : (1) and (2) cannot match because we get the substitutions

$x \rightarrow x-1$ and $0 \rightarrow x$ which are contradictory. We generalize accordingly to

$$g(x,y) = f(x)+y = (1') \Leftarrow (\text{if } x=0 \text{ then } 0 \text{ else } f(x-1)+x) + y.$$

Now, WE APPLY THE FUNCTION $+y$ to the definition of $f(x)$,

$$f(x)+y = (1') \Leftarrow \text{if } x=0 \text{ then } 0+y \text{ else } \underbrace{(f(x-1)+x)+y}_{(2')}$$

The match of (1') and (2') now succeeds with $x \rightarrow x-1, y \rightarrow x+y$ by using the distributivity of $+$.

This proves that $g(x,y) = f(x)+y = \text{if } x=0 \text{ then } y \text{ else } g(x-1,x+y)$, from which we conclude that if we define a functional T by

$T[h](x,y) = \text{if } x=0 \text{ then } y \text{ else } h(x-1,x+y)$ then g is a fixed point of T . We can even prove that g is the least fixed point of T : g is defined on the same domain as f , provided $\omega + y = \omega$ is implicit in the definition of $+$.

$g(x,y) \Leftarrow \text{if } x=0 \text{ then } y \text{ else } g(x-1,x+y)$ and $f(x)+0 = g(x,0) = f(x)$ define a new tail-recursive function, equivalent in the strong sense to $f(x)$.

3.2 Restrictions

The more general scheme

$f(x) \Leftarrow \text{if } a(x) \text{ then } b(x) \text{ else } h(f(e(x)),c(x))$ shows that we need of course a4 and

a5 like WEGBREIT but, instead of a1, a2, a3 we need

a'1: there exists a constant k (not necessarily belonging to the domain of x) such that $h(x, k) = x$ for all x of the domain. This might seem a strong assumption since, for example, the LISP function CONS has no such neutral element. In fact, one has then to use a concatenation function instead of CONS.

3.3 Extensions.

3.3.1. (Shared by WEGBREIT).

The scheme can read

$f(x) \Leftarrow$ if $a(x)$ then $b(x)$ else $h_1(h_2(\dots(h_n(f(e(x), c(x))))))$ provided
 $h_1(h_2(\dots(h_n(h_1(h_2(\dots(h_n(x_1, x_2)))))) = h_1(h_2(\dots(h_n(x_1, \dots))))$ which is some kind of generalized associativity we define as an "f-path idempotency" of the sequence $h_1 \dots h_n$.

3.3.2. Functions of several variables.

It is seen that we do not use the fact that $f(x)$ is in one variable. For instance, if there exists a constant k such that $h(x, k) = x$

$$-h(h(x, y), z) = h(x, h(y, z))$$

$$-h(\omega, x) = \omega.$$

then $f(x, y) \Leftarrow$ if $a(x, y)$ then $b(x, y)$ else $h(f(e(x), e'(y)), c(x, y))$ is equivalent to

$f'(x, y) = g(x, y, k)$ where

$$g(x, y, z) \Leftarrow$$
 if $a(x, y)$ then $h(b(x, y), z)$ else $g(e(x), e'(y), h(c(x, y), z))$.

This is readily seen by a straightforward application of our methodology.

3.3.3. $h(x)$ is an if...then...else... function.

We have therefore a scheme of the type:

$$f(x) \Leftarrow$$
 if $a(x)$ then $b'(x)$ else if $a'(x)$ then $h'(f(e'(x)), c'(x))$
else $h(f(e(x)), c(x))$.

If there exists two constants k and k' such that $h(x, k) = x$ and $h'(x, k') = x$

$$-h(h'(h(h'(x, y), z), t), u) = h(h'(x, h''(y, z, t, u)), h''(y, z, t, u))$$

$$-h(\omega, x) = \omega, h'(\omega, x) = \omega.$$

then $f(x)$ is equivalent to

$f'(x) = g(x, k', k) = h(h'(f(x), k'), k)$ where

$$g(x, z, y) \Leftarrow$$
 if $a(x)$ then $h(h'(b'(x), z), y)$ else if $a'(x)$ then $g(e'(x), h''(c'(x), k, z, y),$
 $h''(c'(x), k, z, y))$
else $g(e(x), h''(k', c(x), z, y),$
 $h''(k', c(x), z, y))$.

This can be finitely iterated.

Finally, we can easily prove that if h and h' are composition functions h_1, \dots, h_n

and h'_1, \dots, h'_n , all of them having a neutral element and $h_1, \dots, h_n, h'_1, \dots, h'_n$ is an "f-path idempotent" sequence then the same methodology applies.

BIBLIOGRAPHY.

- [1] J.ARSAC, Y.KODRATOFF "Some methods for transformation of recursive procedures into iterative one", Communication at the WG2.3 meeting, Zakopane, Jan 79.
- [2] R.M.BURSTALL, J.DARLINGTON, "A transformation system for developing recursive programs", J.ACM 24, 1, 44-67.
- [3] J.DARLINGTON "Program transformation and synthesis : present capabilities", Res; Rep. 77/43, Imperial College or Rep. 48 Dep. of A.I., Edinburgh.
- [4] Z.MANNA, S.NESS, J.VUILLEMIN "Induction methods for proving properties of programs", Comm. ACM, 1973, 16, 491-502.
- [5] J.Mc CARTHY, "Recursive functions of symbolic expressions and their computation by machine", C. ACM 1960, 3, 184-204.