Extending The Benefits Of Tail Recursion

Abstract

Functional programming language provides a much more verifiable language than than iterative programming, particularly if the latter is aimed at efficiency. To extend the use of functional programming it seems to me that deeper understanding of its relation to efficient iterative coding is necessary (Paull 1988). Some simple results toward that end are presented here

A function definition that is ***tail recursive*** can be implemented with reduced stacking cost. Futhermore it can be implemented by its iterative equivalent ***while*** construct completely eliminating stacking with consequent saving in space and time. Either a compiler or programmer can produce the equivalent *while*. Discussion of this fact appears in almost all Programming Language books, from that by R. Sethi to that by M.Scott.

Recursive function definitions which, are not *tail recursive* when directly implemented require stacking, their self calls and unstacking for evaluation. However some such definitions have *tail recursive* equivalents. In this paper I describe some easily recognized non-tail recursive function definition schema with this desireable property.

The defining characteristic of a *tail recursive* definition is that no basic operations are applied outside the arguments of the recursive function being defined.

Tail Recursion And Its Equivalent *while*

We consider tail recursive definitions whose recursive line has one of the two forms :

form 1: $g(F, X) = g(F \$ pX, oX)$: (if) $X \neq X_T$ (X_T is the terminal value of X)
$g(F, X) = F \$ C$: (if) $X == X_T$

form 2: $g(F, X) = g(oX, pX \$ F)$: $X \neq X_T$ && $cond_j(X)$
$g(F, X) = C \$ F$: $X == X_T$

These forms apply if ***F*** and ***X*** are single variables, ***p*** *and* ***o*** are unary functions applied to ***X*** and ***$*** is a binary function applied to ***F*** and ***pX***.

(The forms also apply to the more general situation in which ***X*** represents a set ofvariables, X_1 to X_n, and ***F*** ialso represents set of variables F_1 to F_m .and ***oX*** represents the function(s) o_1 **to** o_n , each applied to subsets of variable(s) in *X* which yields a new set of values of the same variables in *X,* ***pX*** represents the function(s) p_1 to p_m *each* applied to a subset ofthe variable(s) *X* also yielding a new set of values for each of the variables in *X*. ***.*** ***$*** is a set of *m* binary function(s) applied to F_1 to F_m and correspondingly to $p_1 X$ to $p_m X$ to give a resultant set of values to the variables F_1 to F_m.)

Consider a tail recursive line of form 1 above.(Similar conclusions apply for form 2)

Tail-1

$F = 0_{\$}$ $[\ 0_{\$}$ *is 0 in relation to the the operation \$, that is* $F \$ 0_{\$} = F\ ]$

$X = X_0$

$g(F, X) = g(F \$ pX,\ oX)$ $:X != X_T$

$g(F, X) = F \$ C$ $:X == X_T$

The sequence of calls to directly implement this definition are pushed into a runtime stack and evaluated as they are entered so that the final value is in the final entry in the stack. These effectively carry out the following evaluation sequence. (In place of a formal inductive argument the evaluation is carried out for 5 iterations-from which the general case should be clear.)

Eval (Tail-1)

$$g(F_0, X_0) = g([F_0 \$ pX_0],\ o^1X_0)$$
$$= g([[F_0 \$ pX_0] \$ po^1X_0],\ o^2X_0)$$
$$= g([[[F_0 \$ pX_0] \$ po^1X_0] \$ po^2X_0],\ o^3X_0)$$
$$= g([[[[F_0 \$ pX_0] \$ po^1X_0] \$ po^2X_0] \$ po^3X_0],\ o^4X_0)$$
$$= g([[[[[F_0 \$ pX_0] \$ po^1X_0] \$ po^2X_0] \$ po^3X_0] \$ po^4X_0],\ X_T) \quad \text{if } o^5X_0 == X_T$$
$$= [[[[pX_0 \$ po^1X_0] \$ po^2X_0] \$ po^3X_0] \$ po^4X_0] \$ C$$

and if $\$$ is associative

$$g(F_0, X_0) = pX_0 \$ po^1X_0 \$ po^2X_0 \$ po^3X_0 \$ po^4X_0 \$ C$$

The same evaluation can be carried out without the stack by the following iterative construct based on the ***while*** operation. The values that would be put in the stack are now assigned cumulatively to F.

$F = 0_{\$}$

$X = X_0$

$while(X != X_T)$

$\{ F = F \$ pX;$

$X = oX; \}$

$F = F \$ C$

Tracing the genesis of the ***F*** parameter when $o^5X_0 == X_T$

$F = 0_{\$} \$ pX_0 = pX_0$

$X = o^1X_0$

$F = F \$ pX_0 = [pX_0 \$ po^1X_0]$

$X = o^2X$

$F = F \$ pX = [[pX_0 \$ po^1X_0] \$ po^2X_0]$

$\vdots$

$F = [[[[[F_0 \$ pX_0] \$ po^1X_0] \$ o^2X_0] \$ po^3X_0] \$ po^4X_0],\ X_T) \quad \text{if } o^5X_0 == X_T$

$F = [[[[pX_0 \$ po^1X_0] \$ po^2X_0] \$ po^3X_0] \$ po^4X_0] \$ C$

The resultant *while* for a tail recursive definition of form 2 is analogous. It will be detailed later.

In summary, *tail recursive* functions can be evaluated by the function calling itself recursively until the terminal condition is met with the stack returning the computed value in the last call. After the last call there is no need to traipse back through (or rewind) the stack, or, in fact, even to have done the stacking. Alternatively one stack frame can be overwritten with what would have otherwise been the contents of the next stack frame in an iterative, *while*, evaluation.

Equivalence of Non-Tail and Tail Recursion

There are many recursive definitions which can be represented d in a non-*tail recursive* form which also can be represented in *tail recursive* form. Scott (Scott 2006) Notes the fact that the Fibonacci function:

f(N) = f(N-1) + f(N-2) : N > 1
f(N) = 1 : N = 0 or 1

has an equivalent *tail recursive* form. He gives an equivalent *tail recursive* version. More generally any non-tail recursive function of the form:

f(N) = f(N-1) +.. + f((N-j)+... +f(N-n) : N > 1
f(N) = constant : N = 0 or 1

has an equivalent *tail recursive* definition and therefore a simple *while* representation. Here however, we will consider simpler, and perhaps more common forms of non-*tail recursive* definitions which have equivalent *tail recursive* and therefore simple *while* versions.

Simple Transformations From Form 1 Non-Tail To Tail Recursion and Equivalent *while*

Transformation 1

In the tail recursive form there is one occurrence of the function being defined on the left and one on the right and all computation occurs in the arguments of those functions. The following non-*tail recursive* schema has the first of these properties but not the second

$X=X_0$
$f(X) = pX \$ f(oX)$ $:X != X_T$
$f(X) = C$ $:X == X_T$

For illustration, consider the evaluation of ***f(X)*** with the assumption that $o^5X_0 = X_T$, so $f(o^5X_0) = C$

$f(X_0) = [pX_0 \$ f(o^1X_0)]$
$= [pX_0 \$ [po^1X_0 \$ f(o^2X_0)]]$
$= [pX_0 \$ [po^1X_0 \$ [po^2X_0 \$ f(o^3X_0)]]]$
$= [pX_0 \$ [po^1X_0 \$ [po^2X_0 \$ [po^3X_0 \$ f(o^4X_0)]]]]$
$= [pX_0 \$ [po^1X_0 \$ [po^2X_0 \$ [po^3X_0 \$ [po^4X_0 \$ f(o^5X_0 == X_T)]]]]]$
$= [pX_0 \$ [po^1X_0 \$ [po^2X_0 \$ [po^3X_0 \$ [po^4X_0 \$ C)]]]]]$ since $f(X_T) == C$

Therefore

$f(X_0) = g(F_0 , X_0)$, $g(F_0 , X_0)$ being the *tail-recursive* function defined in ***Tail-1*** and whose expansion is found under the heading ***Eval (Tail-1)*** on page 2,

Summarizing

Tail-1

$F = F_0 = 0\$$

$X = X_0 = \langle x_{01},..,x_{0n} \rangle$

$g(F, X) = g(F \$ pX, oX)$ $:X \neq X_T$

$g(F, X) = F \$ C$ $:X == X_T$

is Equivalent to

$X = X_0$

$f(X) = pX \$ f(oX)$ $:X \neq X_T$

$f(X) = C$ $:X == X_T$)

And so can be implemented with the equivalent ***while***

$F = F_0 = 0\$$

$X = X_0$

while $(oX \neq X_T)$

$\{ F = F \$ pX$

$X = oX \}$

$F = C \$ F$

Given:

$L = [L_1, L_2, . . , L_{n-1}, L_n]$

carL is L_1

cdrL is $[L_2, . . , L_{n-1}, L_n]$

listX is $[X]$

|| is concatenation of lists

It is associative

$x \mid L$ is a list with first member x followed by the elements of L. (It is ***cons*** in LISP)*

EXAMPLE

List whose Elements are the Sum of those of 2 given Lists

L1, L2 are lists;

a) $f(L1, L2) = list(carL1 + carL2) \,||\, f(\langle cdrL1, cdrL2 \rangle)$ $:L1 \neq []$

b) $f(L1, L2) = []$ $:L1 == []$

L1, L2 are lists; F=[]

a) $g(F, L1, L2) = g(F \,||\, list(carL1 + carL2) \,||\, cdrL1, cdrL2)$ $:L1 \neq []$

b) $g(F, L1, L2) = F \,||\, []$ $:L1 == []$

and so can be realized by

L1, L2 are lists; F = []

while $(L \neq [])$

$\{ F = F \,||\, list(carL1 + carL2)$

$L1 = cdrL1;$

$L2 = cdrL2; \}$

$F = F \,||\, [] = F$

*The following does **not** have a *tail-recursive* equivalent.

L1 and L2 are lists

$f(L1, L2) = carL1 \mid f(cdrL1, L2)$ $:L \neq []$

$f(L) = L2$ $:L1 == []$

Transformation 2

There is another related recursive definition form which, though not *tail recursive,* also has a *tail-recursive* equivalent and therefore a ***while*** implementation:

$X = X_0$

$f(X) = f(oX) \$ pX$ $:X \neq X_T$

$f(X) = C$ $:X == X_T$

In evaluating this definition, as previously, a sequence of calls monitored by a runtime stack would be executed. These effectively carry out the sequence of evaluations illustrated for five steps below

$f(X_0) = [f(o^1X_0)] \$ pX_0$

$= [[f(o^2X_0)] \$ po^1X_0] \$ pX_0$

$= [[[f(o^3X_0)] \$ po^2X_0] \$ po^1X_0] \$ pX_0$

$= [[[[f(o^4X_0)] \$ po^3X_0] \$ po^2X_0] \$ po^1X_0] \$ pX_0$

$= [[[[[f(o^5X_0==X_t)] \$ po^4X_0] \$ po^3X_0)] \$ po^2X_0] \$ po^1X_0] \$ pX_0$

$f(X_0) = [[[[C \$ po^4X_0] \$ po^3X_0] \$ po^2X_0] \$ po^1X_0] \$ pX_0$ *if* $o^5X_0==X_T$

if $ is **associative**

$f(X) = C \$ po^4X_0 \$ po^3X_o \$ po^2X_0 \$ po^1X_0 \$ pX$

This then is equivalent to the evaluation for

Tail-2

$F = F_0 = 0\$$ *[$0_\$$ is 0 in relation to the the operation $, that is* $0_\$ \$ F = F$ *]*

$X = X_0$

$g(F, X) = g(pX \$ F, oX)$ *:X !=* X_T

$g(F, X) = C \$ F$ *:X ==* X_T

which when evaluated gives\

val (Tail-2)

$g(F_0, X_0) = g(pX_0, o^1X_0) = g([pX_0], o^1X) =$

$= g([po^1X_0 \$ pX_0], o^2X_0)$

$= g([po^2X_0 \$ [po^1X_0 \$ pX_0]], o^3X_0)$

$= g([po^3X_0 \$ [po^2X_0 \$ [po^1X_0 \$ pX_0]]], o^4X_0)$

$= g([po^4X_0 \$ [po^3X_0 \$ [po^2X_0 \$ [poX_0 \$ pX_0]]]], o^5X_0)$

$= C + [po^4X_0 \$ [po^3X_0 \$ [po^2X_0 \$ [poX_0 \$ pX_0]]]]$ *if* $o^5X_0==X_T$

If $ is **associative:**

$g(F_0, X_0) = C \$ po^4X_0 \$ po^3X_0 \$ po^2X_0 \$ poX_0 \$ pX_0$ Assuming $o^5X_0==X_t$

And this tail recursive function is equivalent to the following ***while***

$F = F_0 = 0\$$

$X = X_0$

while $(X != X_t)$

{ $F = pX \$ F$

$X = oX$ *}*

$F = C \$ F$

EXAMPLE: **Reverse a List**

Given:

$L = [L_1, L_2, \ldots, L_{n-1}, L_n]$

carL is L_1

cdrL is $<L_2, \ldots, L_{n-1}, L_n>$

listX is $<X>$

|| is concatenation of lists

It is associative

[] is the empty list

L = a list

a) f(L) = f(cdrL)|| list(carL) :L != []
b) f(L) = [] :L == []

is equivalent to

L = a list; F=[]

a) g(F, L) = g(list(carL) || F, cdrL) :L != []
b) g(F,L) =[] || F :L == []

and so can be realized by

```
L = a list; F=[]
while(L != [] )
 { F = list(carL )||  F  )
   L = cdrX; }
F = [] || F = F
```

The following example are cases in which *$* is both associative and commutative. Therefore they can be expressed in either of the two forms considered previously, and implemented by either of the two forms of tail recursive and their corresponding *while.*

EXAMPLE

Integer Division

R and D are integers
$R = R_0$ *and* $0 =< D =< R_0$

a) f(R,D) = 1 + f(R-D, D) :R >= D
b f(R,D) = 0_+ *:R < D*

is equivalent to

$F = 0 = 0_+$; $R = R_0$ *and* $0 =< D =< R_0$

a) g(F,<R,D>) = g(F+1,< R-D, D> :R >= D
b) g(F,<R,D>) = F :R < D

and so can be realized by

$F = 0 = 0_+$; $R = R_0$ *and* $0 =< D =< R_0$

```
while(R >= D )
 { F = F + 1;
   R = R - D; }
```

Sum Of Components Of An Array

X is an array X[1...N], I = index, $I_0 = 1$

a) f(X, N, I) = X[I] + f(X, I+1,N) : i =< N
b) f(X, N, I) = 0 : i = N+1

is equivalent to

F = 0;

a) g(F, <X, I, N>) = g(F+X[I],<X, I+1, N>) :I =< N
b) g(F, <X, I, N>) = F :I == N+1

and so can be realized by

I =< N
$X = X_0$ *is an array X[1...N]; I =* $I_0 = 1$

```
while(I =< N )
 {  F =  F +  X[I];
    I = I + 1;  }
```

Summary

	Recursive Line Of Function Definition	Expanded Form Of Function Assuming $o^5X_0 == X_t$
1	$f(X) = pX \$ f(oX)$	$f(X_0) = pX_0 \$ [po^1X_0 \$ [po^2X_0 \$ [po^3X_0 \$ po^4X_0]]]$
2	$g(F, X) = g(F \$ pX, oX)$	$g(F_0, X_0) = [[[pX_0 \$ po^1X_0] \$ po^2X_0] \$ po^3X_0] \$ po^4X_0$
3	$f(X) = f(oX) \$ pX$	$f(X_0) = [[[po^4X_0 \$ po^3X_0] \$ po^2X_0] \$ po^1X_0] \$ pX_0$
4	$g(F, X) = g(pX \$ F, oX)$	$g(F_0, X_0) = po^4X_0 \$ [po^3X_0 \$ [po^2X_0 \$ [poX_0 \$ pX_0]]]$
If *$* is associative *1* is equivalent to *2*, and *3* is equivalent to *4.* *2* and *4* have equivalent ***whiles***		

Other Equivalences

Through the property of association of ***$*** the equivalence between non-*tail recursive* and *tail recursive* functions has been established. We note that even in the absence of this property an equivalence can be established if ***$*** has other simple properties.

Transformation 3

Earlier, the evaluation of

$X=X_0$
$f(X) = pX \$ f(oX)$ $:X != X_T$
$f(X) = C$ $:X == X_T$

was given under the asummption that

$o^5X_0 = X_T$, so $f(o^5X_0) = C$

Now, unlike our previous assumption, suppose that ***$*** is not associative, but the inverse of $o^1(X)$, $o^{-1}(X)$, as well as X_0 and X_T are all known. Under these conditions the function equivalent to ***f(X)*** in *tail recursive* form can be formulated. For the example above (assuming that $o^5X_0 = X_T$, so $f(o^5X_0) = C$ and the resultant tail recursive definition is:

$F = C$
$X = X_T$
$g(F, X) = g(o^{-1}X,\ p\ o^{-1}X \$ F)$ $:X != X_T$
$g(F, X) = C \$ F$ $:X == X_T$

Evaluating this definition under the stated conditions gives:

$$g(o^{-1}(o^5X_0),\ po^{-1}(o^5X_0) \$ C) = g(o^4X_0,\ [po^4X_0 \$ C])$$
$$= g(o^3X_0,\ [po^3X_0 \$ [po^4X_0 \$ C]])$$
$$= g(o^2X_0,\ [po^2X_0 \$ [po^3X_0 \$ [po^4X_0 \$ C]]])$$
$$= g(o^1X_0,\ [po^1X_0 \$ [p\ o^2X_0 \$ [po^3X_0 \$ [po^4X_0 \$ C]]]])$$
$$= g(X_0,\ [pX_0 \$ [po^1X_0 \$ [p\ o^2X_0 \$ [po^3X_0 \$ [po^4X_0 \$ C]]]]])$$
$$= [pX_0 \$ [po^1X_0 \$ [p\ o^2X_0 \$ [po^3X_0 \$ [po^4X_0 \$ C]]]]$$

So the equivalent ***while*** is:

$F = C$
$X = X_T$
$while(X != X_0)$
$\{ X = o^{-1}X$
$F = pX \$ F \}$

Though the properties necessary to implement a more efficient *tail-recursive* definition, given the recursive one above, do not hold. The reason it doesn't may be of interest. A reasonable change in the underlying data-structure may make those properties hold. Consider for example the definition given earlier as one

that uses the non-associative | (cons) function and does not have a non-*tail-recursive equivalent* .

L1 and L2 are lists
f(L1, L2) = carL1 | f(cdrL1, L2) :L1 != []
f(L) =L2 :L1 == []

One needs the last member of L1 in order to begin using the inverse o^{-1} . However that inverse is not immediately available without building a function to give the last member of a list. If one could get that last member quickly because perhaps there is a pointer to the end of a list as well its beginning the implementation of ***f(L1,L2)*** would run significantly faster.

Including Multiple Conditions in Definitions

The transformation from non-*tail recursive* to *tail recursive* definitions with consequent increased in efficiency can be extended to definitions which incorporate different equalities under different conditions. Below the forms of the non-*tail recursive* definitions and the equivalent *tail recursive f*unction which is available if the appropriate conditions of associativity holds. Finally the equivalent *while* construct is given. As noted In the *while* construct the tests and resultant assignments are assumed to be in parallel. Therefore the conditions are assumed, as in the recursive forms, to be mutually exclusive.

Non-Tail Recursive

$X=X_0$

$f(X) = p_1X \$ f(oX)$: $X != X_T$ & $cond1(X)$

$f(X) = p_nX \$ f(oX)$: $X== X_T$ & $cond1(X)$

$f(X) = C_1$: $X == X_T$ & $condT_1(X))$

$f(X) = C_1$: $X == X_T$ & $condT_m(X))$

Equivalent Tail Recursive

$F = F_0$

$X = X_0$

$g(F, X) = g(F \$ p_1X , oX)$: $X != X_T$ & $cond1(X)$

$g(F, X) = g(F \$ p_nX , oX)$: $X != X_T$ & $condn(X)$

$g(F, X) = t_1(F) \};$: $X == X_T$ & $condT_1(X))$

$g(F, X)= t_m(F) \};$: $X == X_T$ & $condT_m(X))$

$cond_1(X)$ to $cond_n(X)$ are mutually exclusive

$condT_1(X)$ to $condT_m(X)$ are mutually exclusive

Equivalent *while*

while(X != X_T)

{ if (cond$_1$(X) { F = F $ p_1X; X = o_1X; }

⋮

if (cond$_n$(X)) { F = F $ p_nX; X = o_nX; }

}

if (condT$_1$(X)) { F = t_1(F) };

⋮

if (condT$_m$(X)) { F = t_m(F) };

cond$_1$(X) to ***cond$_n$(X)*** are mutually exclusive it is assumed that ***{ F = F $ p_jX; X = o_jX; }*** *is* a **parallel** specification. We could add ***F*** to the argument of ***o_j*** but the same effect can be obtained, albeit inefficiently, by extending ***X*** to include ***F.***

condT$_1$(X) to ***condT$_m$(X)*** are mutually exclusive

Conclusion

A class of simple non-*tail recursive* functions, F, has been shown to have equivalent *tail recursive* and iterative *while* representations. So the efficiency of their implementations can be significantly improved. F is defined by the simple form and the properties of the basic operation, *o*, which its members employ. In order for the equivalence to exist this *o* must be associative. If it is not associative the equivalence could still hold if it had a number of other efficiently implemented properties, including an inverse. If, as implemented, these properties were not efficiently implemented, a change in the representation (the underlying data-structure) on which *o* depends may make the property efficient.

REFERENCES

1. PAULL, MARVIN C.(1988): *Algorithm Design A Recursion Transformation Framework,* New York: Wiley-Interscience,.

2. SCOTT, MICHAEL L.(2006): *Pragmatics Of Programming Languages ,2nd edition,* Morgan Kauffman , pp 298, 623.

3. SETHI, RAVI (1977):*Programming Languages, Concepts and Constructs, 2nd edition*, Reading, Mass , Addison Wesley, pp186,190.