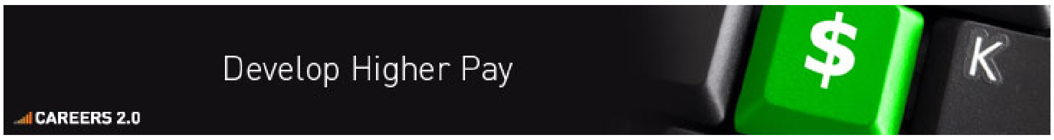
Can all iterative algorithms be expressed recursively?



If not, is there a good counter example that shows an iterative algorithm for which there exists no recursive counterpart?

If it is the case that all iterative algorithms can be expressed recursively, are there cases in which this is more difficult to do?

Also, what role does the programming language play in all this? I can imagine that Scheme programmers have a different take on iteration (= tail-recursion) and stack usage than Java-only programmers.

programming-languages recursion iteration language-theory

asked Jan 19 '10 at 13:08
eljenso
7,899 3 30 50

1 mathoverflow.com – jldupont Jan 19 '10 at 13:10

add a comment

7 Answers

There's a simple ad hoc proof for this. Since you can build a Turing complete language using strictly iterative structures and a Turning complete language using only recursive structures, then the two are therefore equivalent.

answered Jan 19 '10 at 13:20
plinth
27.8k 6 44 88

1 Wow, food for thought. I envy all the upvoters that digested this more quickly than I did. Time to read up on this. – eljenso Jan 19 '10 at 13:53

Wait, isn't that a logical fallacy? Circular reasoning? – C Bauer Jan 19 '10 at 14:43

7 C Bauer: Actually, it is not. The proof is very easy to do: assume languages IT (with iterative constructs only) and REC (with recursive constructs only). Simulate a universal turing machine using IT, then simulate a universal turing machine using REC. The existence of the simulator programs guarantees that both IT and REC can calculate all the computable functions. This property is proven for the lambda calculus, where all functions are partial recursive. – fishlips Jan 19 '10 at 16:18

1 Alternately, build a Turing complete language using strictly (iteration/recursion), and in it write an interpreter for a Turing complete language using strictly (recursion/iteration). Voila, whatever program you've written is being executed iteratively and recursively, simultaneously! – David Thornley Jan 19 '10 at 21:13

1 The appeal to higher authority... – Norman Ramsey Jan 22 '11 at 18:18

add a comment

Find your next developer with Careers 2.0.
It's like finding hay in a haystack.
CAREERS 2.0

> Can all iterative algorithms be expressed recursively?

Yes, but the proof is not interesting:

1. Transform the program with all its control flow into a single loop containing a single case statement in which each branch is straight-line control flow possibly including `break` , `return` , `exit` ,

`raise` , and so on. Introduce a new variable (call it the "program counter") which the case statement uses to decide which block to execute next.

This construction was discovered during the great "structured-programming wars" of the 1960s when people were arguing the relative expressive power of various control-flow constructs.

2. Replace the loop with a recursive function, and replace every mutable local variable with a parameter to that function. Voilà! Iteration replaced by recursion.

This procedure amounts to writing an interpreter for the original function. As you may imagine, it results in unreadable code, and it is not an interesting thing to do. *However*, some of the techniques can be useful for a person with background in imperative programming who is learning to program in a functional language for the first time.

answered Jan 19 '10 at 21:08
Norman Ramsey
119k 30 230 421

Aw, @Norman, it is an interesting thing to do.. for a compiler. In fact the procedure is: transform imperative code to functional code, then, transform functional code to imperative code. Why is this interesting? Because the functional code has simple semantics but can't be executed, whereas the imperative output is incomprehensible but suitable for execution. In particular the functional code is easy to optimise for high level things and the imperative code for low level things (but the initial mix is hard to work with for any purpose). – Yttrill Jan 22 '11 at 14:55

add a comment

Like you say, every iterative approach can be turned into a "recursive" one, and with tail calls, the stack will not explode either. :-) In fact, that's actually how Scheme implements all common forms of looping. Example in Scheme:

```
(define (fib n)
  (do ((x 0 y)
       (y 1 (+ x y))
       (i 1 (+ i 1)))
      ((> i n) x)))
```

Here, although the function looks iterative, it actually recurses on an internal lambda that takes three parameters, `x` , `y` , and `i` , and calling itself with new values at each iteration.

Here's one way that function could be macro-expanded:

```
(define (fib n)
  (letrec ((inner (lambda (x y i)
                    (if (> i n) x
                        (inner y (+ x y) (+ i 1))))))
    (inner 0 1 1)))
```

This way, the recursive nature becomes more visually apparent.

answered Jan 19 '10 at 13:22
Chris Jester-Young
109k 20 213 293

3 And do note that *any* iterative algorithm can be turned into a tail-recursive algorithm. For instance, just transform it to continuation-passing style. – Derrick Turk Jan 19 '10 at 13:24

I would just add that not every language compiler optimizes tail calls, so the stack can indeed "explode" (overflow) in those languages using tail recursion (ex. C#). – dcstraw Mar 9 '10 at 16:56

add a comment

Defining iterative as:

```
function q(vars):
  while X:
    do Y
```

Can be translated as:

```
 function q(vars):
    if X:
      do Y
      call q(vars)
```

Y in most cases would include incrementing a counter that is tested by X. This variable will have to be passed along in 'vars' in some way when going the recursive route.

answered Jan 19 '10 at 13:23
Johan
1,669 9 13

add a comment

As noted by plinth in the their answer we can construct proofs showing how recursion and iteration are equivalent and can both be used to solve the same problem; however, even though we know the two are equivalent there are drawbacks to use one over the other.

In languages that are not optimized for recursion you may find that an algorithm using iteration preforms faster than the recursive one and likewise, even in optimized languages you may find that an algorithm using iteration written in a different language runs faster than the recursive one. Furthermore, there may not be an obvious way of written a given algorithm using recursion versus iteration and vice versa. This can lead to code that is difficult to read which leads to maintainability issues.

answered Jan 19 '10 at 13:46
SecretSquirrel
9,164 7 52 91

add a comment

Prolog is recursive only language and you can do pretty much everything in it (I don't suggest you do, but you can :))

answered Jan 19 '10 at 13:24
dmonlord
1,058 6 13

add a comment

Recursive Solutions are usually **relatively inefficient** when compared to iterative solutions. However, it is noted that there are some problems that can be solved only through recursion and equivalent iterative solution may not exist or extremely complex to program easily (Example of such is **The Ackermann function** cannot be expressed without recursion)Though recursions are elegant,easy to write and understand.

answered Nov 14 '13 at 1:53
yogi
1

"Ackermann function cannot be expressed without recursion" This is not true. How do you think recursion is implemented in a computer? The CPU operates iteratively on a sequence of instructions. To support function calls, including recursive calls, it manages a *stack*. Using recursion is simply letting the *language* (OS, runtime, etc.) manage a stack for you. Any recursive algorithm can be replaced with an iterative one where you manage the stack yourself, including Ackermann. – Mud Aug 14 at 19:42

add a comment

Not the answer you're looking for? Browse other questions tagged programming-languages recursion iteration language-theory or ask your own question.