Last time...

- Recursive Correctness
- Karatsuba's Algorithm

CSC 236 Lecture 8: Correctness 2

Harry Sha

July 12, 2023

Today

Correctness for Iterative Algorithms

Correctness of `merge`

Correctness for Iterative Algorithms

Correctness of `merge`

Iterative Algorithms

Iterative Algorithms are algorithms with a `for` loop or a `while` loop in them.

Conventions

```
i = 0
while i < N:
    ...
    i += 1
```

- "After the kth iteration" refers to the point in the execution of the program just before the loop condition is evaluated for the $k+1$st time.
- "Before the $k+1$th iteration" is the exact same thing as "after the kth iteration"
- For example, after the kth iteration, the value of i is k

A multiplication algorithm for natural numbers

```
def mult(x, y):
    i = 0
    total = 0
    while i < x:
        total = total + y
        i += 1
    return total
```

What is the pre/post condition for `mult`(x, y)?

PRE $\cancel{n}$ $x \in \mathbb{N}$, $y \in \mathbb{R}$.

POST: returns $x \cdot y$

Proof

Define $P(n)$: after the nth iteration, · total $= ny$

· $i = n$

WTS $\forall n \in \mathbb{N}, P(n)$.

Base case: $P(0)$: total $= 0 = 0 \cdot y$ ✓, $i = 0 = 0$ ✓

Inductive step: Suppose $P(k)$ WTS $P(k+1)$.

if the $k+1$th iteration did not run, $P(k+1)$ is vacuously true. Otherwise,

$i_{k+1} = i_k + 1$

$= k+1$

$$\begin{aligned} \text{total}_{k+1} &= \text{total}_k + y \\ &= ky + y \\ &= (k+1)y \end{aligned}$$

This completes the induction.

for $j < x$, $P(j) \Rightarrow$ $i_j = j < x$, so every iteration before x runs,

$P(x) \Rightarrow i_x = x$ so the loop condition fails.

$\Rightarrow$ total $= xy$.

Convention

Use subscripts to denote the value of a variable after iteration i.
E.g., $\texttt{total}_i$ is the value of the variable `total` after iteration i.

General Strategy

Define a loop invariant - some property that is true at the end of every iteration. Note that it can depend on the iteration number. Call it, for example, $P(n)$. Another common one is $P(i)$ if you use i as the iteration counter.

Tip: Since the value of variables in code can change at each iteration, it is useful to use the convention in the previous slide to refer to the value of a variable after a certain iteration.

General Strategy

Prove the following:

Base case

Initialization. Show that the loop invariant is true at the start of the loop if the precondition holds.

Inductive step.

Maintenance. Show that if the loop invariant is true at the start of any iteration, it is also true at the start of the next iteration.

Termination. Show that the loop terminates and that when the loop terminates, the loop invariant applied to the last iteration implies the postcondition.

Runtime?

```python
def mult(x, y):
    i = 0
    total = 0
    while i < x:
        total = total + y
        i += 1
    return total
```

$\underbrace{999\ldots9}_{n}$ $10^n - 1$

$= \Theta(10^n)$

$\boxed{\Theta(n \cdot 10^n)}$

GS: $\Theta(n^2)$

K: $O(n^{1.59})$ Best$(n\log(n))$

Let's say x and y are both n-**digit** numbers and it takes time $O(n)$ time to add two n digit numbers.

What is the worst-case time complexity of `mult` in terms of n, the number of digits?

Runtime?

Since y is a n digit number, it can be as large as 999...99 (n-times), which is equal to $10^n - 1 = \Theta(10^n)$. Thus, the loop runs for $O(10^n)$ iterations!

The eventual result has as many $2n$ digits. Thus, each addition takes time $O(2n) = O(n)$. In total, the running time is $\Theta(n10^n)$.

This is terrible. For reference, Grade School Multiplication gets $O(n^2)$, and Karatsuba's Algorithm from last week gets $O(n^{1.59})$. The best-known algorithm for multiplying has runtime $O(n \log(n))$. By the way, this fast algorithm was just discovered in 2019 and published in 2021!

`for` Loops

`for` loops are another type of loop. You can think of loops as while loops with an appropriate loop condition. For example

```
for i in range(0, 10):
    ...


# is the same as


i = 0
while i < 10:
    ...
    i+1
```

Termination

Termination can usually be proved as a consequence of the loop invariant.

Usually, the argument will go something like this.

- By contradiction, suppose the loop didn't terminate. Then it reaches iteration N (where N is some value you chose, big enough to derive a contradiction).
- Then the loop invariant $P(N)$ implies that the value of some variables is something. This implies the loop condition will be false in the next iteration, which is a contradiction.

Termination

If you're more precise, you can often find the exact number of iterations using the Loop Invariant. That looks something like

- Claim: The loop exits after the Nth iteration
- Let $i < N$, $P(i)$ implies that the loop condition is true.
- Furthermore $P(N)$ implies that the loop condition is false. Therefore, the loop ~~exists~~ after the Nth iteration.
  exits

Mystery algorithm

What does the following algorithm do?

- Precondition: $x, y \in \mathbb{N}$, $y > 0$.

Mystery algorithm

What does the following algorithm do?

- Precondition: $x, y \in \mathbb{N}$, $y > 0$.
- Postcondition: Returns $\lceil x/y \rceil$.

```python
def mystery(x, y):
    val = 0
    c = 0
    while val < x:
        val += y
        c += 1
    return c
```

~~val~~ $P(n)$: $val_n = ny$

$c_n = n$

Proof of Correctness

Loop Invariant. $P(n)$ is the following predicate. After the nth iteration

a.) $c = n$

b.) $\texttt{val} = ny$

We'll show $\forall n \in \mathbb{N}.P(n)$

Proof of Correctness

$P(n)$: After the nth iteration

a.) $c = n$

b.) $\texttt{val} = ny$

WTS $\forall n\ P(n)$.

Initialization: $\text{Val}_0 = 0 = 0 \cdot y$

$c_0 = 0 = 0$ ✓

Maintenance: suppose $P(k)$ WTS $P(k+1)$.

$$\text{Val}_{k+1} = \text{Val}_k + y = ky + y = (k+1)y$$

$$c_{k+1} = c_k + 1 = k+1.$$

Termination: By contradiction suppose the loop doesn't terminate, then it must reach iteration $N = \boxed{100xy}$. then $P(N) \Rightarrow \text{val} = 100xy^2 > x$

$\Rightarrow$ the loop terminates which is a contradiction.

Proof of Correctness

$P(n)$: After the nth iteration

a.) $c = n$

b.) $\texttt{val} = ny$

So, the loop terminates.

Let N be the first iteration after which the loop condition fails. Note this means the loop condition is true after the N-1 th condition.

$$Val_{N-1} < x \leq Val_N$$

$$\xRightarrow{P(N),P(N-1)} (N-1)y < x \leq Ny$$

$$(N-1) < \frac{x}{y} \leq N$$

$\Rightarrow$ N is the first natural number $\geq x/y = \lceil x/y \rceil$.

we return $C_N = N = \lceil x/y \rceil$.

Convention

If the predicate $P(n)$ has multiple parts like

a.) ...

b.) ...

Use $P(n)$.a, $P(n)$.b,... to refer to specific parts of the predicate.

Variations

- One after the other. Prove the correctness of each loop in sequence.
- Nested loops. "inside out". Decompose (or imagine) the inner loop as a separate function. Prove the correctness of that function as a lemma, and then prove the correctness of the outer loop. We will see an example in the tutorial.

Another way to prove termination: descending sequence

Another way to prove termination is to define a strictly descending sequence of natural numbers, $a_1, a_2, \ldots$ indexed by the iteration number.

Another way to prove termination: descending sequence

Another way to prove termination is to define a descending sequence of natural numbers, $a_1, a_2, ...$ indexed by the iteration number.

By the WOP, this sequence must be finite; otherwise, the set $\{a_1, a_2, ...\}$ has no minimal element!

Example

$a_{k+1} = X - Val_k - y$

Want: $a_{k+1} \geq 0$

$Val_k < X$

$0 < \underbrace{X - Val_k + y}_{a_{k+1}}$

How can we define a descending sequence of natural numbers for this algorithm?

```python
def mystery(x, y):
    val = 0
    c = 0
    while val < x:
        val += y
        c += 1
    return c
```

need to use loop condition!!

$a_n = X - Val + y$

Claim: $a_1, a_2 \ldots$ is strictly sequence of natural numbers.

By induction: $a_0 = X - 0 = x$

suppose $a_1, \ldots, a_k$ is a descending sequence of natural numbers, we'll show $a_1, \ldots, a_{k+1}$ is also a descending seq of natural numbers.

$\boxed{a_{k+1}} = x - Val_{k+1} + y$

$= X - (k+1) + y = \underbrace{\boxed{X - ky}}_{a_k = X - Val_k} + y.$

$= a_k - y$

Descending Sequence

Proofs of termination: as a part of the LI vs. descending sequence

Most of the time, the LI will imply termination, saving you from having to do another induction proof. I prefer this method.

Proofs of termination: as a part of the LI vs. descending sequence

Most of the time, the LI will imply termination, saving you from having to do another induction proof. I prefer this method.

However, it is easier to define a descending sequence of natural numbers in some cases - we'll see some examples in the tutorial.

Correctness for Iterative Algorithms

Correctness of `merge`

Merge

```python
def merge(x, y):
    l = []
    while len(x) > 0 or len(y) > 0:
        if len(x) > 0 and len(y) > 0:
            if y[0] <= x[0]:
                l.append(y.pop(0)) # 1.
            else:
                l.append(x.pop(0)) # 2.
        elif len(x) == 0:
            l.append(y.pop(0)) # 3.
        else: # len(y) == 0
            l.append(x.pop(0)) # 4.
    return l
```

Correctness of Merge

- Precondition? → x, y are sorted.
- Postcondition? → sorted list

Counters

$$\mathrm{Counter}([1,1,1,2,2,3,4]) = \{1:3,\ 2:2,\ 3:1,\ 4:1\}.$$

For a list l of natural numbers, let `Counter`(l) be a mapping of the elements of l to the number of times they appear. Ways to think about this

- $\boxed{\text{collections.Counter}}$
- `Counter`(l) can be thought of as a multiset (an unordered collection of objects where the same object can appear multiple times)
- `Counter`(l) : $\mathbb{N} \to \mathbb{N}$ where `Counter`(l)(x) is the number of times x appears in l.

Counters

We can use Counters to express the pre and postconditions more formally.

Precondition. x and y are sorted lists of natural numbers.

Postcondition. Returns a sorted list l such that $\texttt{Counter}(l) = \texttt{Counter}(x + y)$, note that the $+$ here is concatenation of lists. This means the returned list is sorted and contains all the elements in x and y with the correct frequencies.

Correctness of `merge`

Loop Invariant.

$P(n)$: After the nth iteration,

a.) $(a \in \boxed{x_n + y_n} \wedge \boxed{b} \in \boxed{l_n}) \implies a \geq b$.

b.) $\texttt{Counter}(x_n + y_n + l_n) = \texttt{Counter}(x_0 + y_0)$.

c.) $\texttt{len}(l_n) = n$.

d.) x_n, y_n, l_n are all sorted.

Correctness of $\texttt{merge}$

$P(n)$: After the nth iteration,

a.) $(a \in x_n + y_n \wedge b \in \underline{l_n}) \implies a \geq b$.

b.) $\texttt{Counter}(x_n + y_n + l_n) = \texttt{Counter}(x_0 + y_0)$.

c.) $\texttt{len}(l_n) = n$.

d.) x_n, y_n, l_n are all sorted.

Initialization : $P(0)$

a) vacuosly true ✓

b) $\text{Counter}(x_0 + y_0 + l_0) = \text{Counter}(x_0 + y_0)$ since $l_0 = []$ ✓

c) $\text{len}([]) = 0$ ✓

d) x_0, y_0 are sorted by PRE, l_0 is sorted b/c it's empty.

Correctness of `merge`

$P(n)$: After the nth iteration,

a.) $(a \in x_n + y_n \wedge b \in l_n) \implies a \geq b$.

b.) $\texttt{Counter}(x_n + y_n + l_n) = \texttt{Counter}(x_0 + y_0)$.

c.) $\texttt{len}(l_n) = n$.

d.) x_n, y_n, l_n are all sorted.

Correctness of `merge`

$P(n)$: After the nth iteration,

a.) $(a \in x_n + y_n \wedge b \in l_n) \implies a \geq b$.

b.) $\texttt{Counter}(x_n + y_n + l_n) = \texttt{Counter}(x_0 + y_0)$.

c.) $\texttt{len}(l_n) = n$.

d.) x_n, y_n, l_n are all sorted.

maintenance: suppose $P(k)$

WTS $P(k+1)$.

By cases: let's look at case 1:

$$l_{k+1} = l_k + y_k[0], \quad y_{k+1} = y[1:]$$

a.) let $a \in x_{k+1} + y_{k+1}$, $b \in l_{k+1}$.

if $a, b \in x_k + y_k, l_k$ respectively, true by $P(k)$.a

else $b = y_k[0]$. in this case.

$b \leq y_k[1:]$ b/c y_k is sorted by $P(k)$.d

$b \leq x_{k+1}$ by the case split $y_k[0] \leq x_k[0]$

and x_k is sorted by $P(k)$.d

Correctness of merge

$P(n)$: After the nth iteration,

a.) $(a \in x_n + y_n \wedge b \in l_n) \implies a \geq b$.

b.) $\texttt{Counter}(x_n + y_n + l_n) = \texttt{Counter}(x_0 + y_0)$.

c.) $\texttt{len}(l_n) = n$.

d.) x_n, y_n, l_n are all sorted.

Case 1 b.) true. b/c we simply moved $y_k[0]$ to l.

$l_{k+1} = l_k + y_k[0]$, $y_{k+1} = y[1:]$

$$\text{Counter}(x_{k+1} + y_{k+1} + l_{k+1}) = \text{Counter}(x_k + y_k[1:] + l_k + y_k[0])$$

$$\xrightarrow{\text{unordered}} = \text{Counter}(x_k + y_k + l_k) = \text{Counter}(x_0 + y_0)$$

c) $\text{len}(l_{k+1}) = \text{len}(l_k + 1) = k+1$ ✓.

the rest of the cases are similar.

d) x_{k+1} is unchanged

y_{k+1} just lost an element

suffices to show that $y_k[0] \geq$ everything in l_k. P(k).a

$\Rightarrow l_{k+1}$ is still sorted.

$P(n)$: After the nth iteration,

a.) $(a \in x_n + y_n \wedge b \in l_n) \implies a \geq b$.

b.) $\texttt{Counter}(x_n + y_n + l_n) = \texttt{Counter}(x_0 + y_0)$.

c.) $\texttt{len}(l_n) = n$.

d.) x_n, y_n, l_n are all sorted.

Termination: let $n = \text{len}(x_0) + \text{len}(y_0)$.

claim: loop terminates after the nth iteration.

indeed $P(n) \Rightarrow$ $\text{len}(l_n) = n$ and,

$\text{Counter}(x_n + y_n + \underbrace{l_n}_{n}) = \text{Counter}(\underbrace{x_0 + y_0}_{n})$.

since $\text{len}(l_n) = n$, x_n, y_n must have length 0. $\Rightarrow$ loop cond fails.

then $P(n).d \Rightarrow$ $\boxed{l_n \text{ sorted}}$ POST.

$P(n).b \Rightarrow$ $\boxed{\text{Counter}(l_n) = \text{Counter}(x_0 + y_0)}$

Loop Invariants

- It's normal for the loop invariant to have many parts!
- If you're trying to prove a loop invariant and you get stuck and wish some other property holds, try adding what you need as part of the loop invariant.
- For example, it's common for part 4. of a loop invariant to imply part 1. of the loop invariant.

Summary - Correctness of Algorithms

- If the algorithm is recursive, prove correctness directly by induction.
- For algorithms with loops, prove the correctness of the loop by defining a Loop Invariant, proving the Loop Invariant, and showing that the Loop Invariant holding at the end of the algorithm implies the postcondition.

What are your questions?