CSC236 Homework Assignment #2

Induction Proofs on Program Correctness and Recurrences

Alexander He Meng

Prepared for October 28, 2024

Question #1

Consider the following program from pg. 53-54 of the course textbook:

```python
def avg(A):
    """
    Pre: A is a non-empty list
    Post: Returns the average of the numbers in A
    """
    sum = 0
    i = 0
    while i < len(A):
        sum += A[i]
        i += 1
    return sum / len(A)

print(avg([1, 2, 3, 4]))  # Example usage
```

Denote the predicate:

$$Q(j) : \text{At the beginning of the } j^{\text{th}} \text{ iteration, } \texttt{sum}_j = \sum_{k=0}^{i_j - 1} A[k].$$

Claim:

$\forall j \in \{1, \ldots, len(A)\}, Q(j)$

Proof.

This proof leverages the Principle of Simple Induction.

Base Case:

Let $j = 1$.

At the beginning of the 1^{st} iteration, $\texttt{sum}_1 = 0$ and $i_1 = 0$.

It follows that

$$\texttt{sum}_1 = \sum_{k=0}^{i_1-1} A[k] = \sum_{k=0}^{0-1} A[k] = \sum_{k=0}^{-1} A[k] = 0.$$

Hence, $Q(1)$.

<u>Induction Hypothesis:</u>
Assume for some iteration $m \in \{1, ..., len(A) - 1\}$, $Q(m)$.

Namely, for the m^{th} iteration,

$$\texttt{sum}_m = \sum_{k=0}^{i_m-1} A[k].$$

<u>Induction Step:</u>
Proceed to show $Q(m+1)$:

Notice that $\texttt{sum}_{m+1} = \texttt{sum}_m + A[i_{m+1}]$, by *Line 9* of the program.

By the Induction Hypothesis,

$$\texttt{sum}_m + A[i_{m+1}] = \sum_{k=0}^{i_m-1} A[k] + A[i_{m+1}],$$

and by *Line 10* of the program, $i_{m+1} = i_m + 1$;

$$\sum_{k=0}^{i_m-1} A[k] + A[i_{m+1}] = \sum_{k=0}^{i_{m+1}-1} A[k].$$

Thus,

$$\texttt{sum}_{m+1} = \sum_{k=0}^{i_{m+1}-1} A[k]$$

as needed.

Therefore, by the Principle of Simple Induction, $Q(j)$ holds for all $j \in \{1, \ldots, len(A)\}$.

$\square$

Question #2

Recall $Q(j)$ from *Question # 1*:

$$Q(j) : \text{At the beginning of the } j^{\text{th}} \text{ iteration, } \texttt{sum}_j = \sum_{k=0}^{i_j - 1} A[k].$$

Denote the following predicate:

$$Q'(n) : 0 \leq n < len(A) \implies Q(n+1)$$

<u>**Claim:**</u>
Proving $\forall j \in \{1, \ldots, len(A)\}, Q(j)$ is equivalent to proving $\forall n \in \mathbb{N}, Q'(n)$.

Proof.
<u>Remarks</u>
It is sufficient to show that $\forall j \in \{1, \ldots, len(A)\}, Q(j) \iff \forall n \in \mathbb{N}, Q'(n)$, to show that proving one of these statements is equivalent to proving the other.

<u>$(\forall j \in \{1, \ldots, len(A)\}, Q(j)) \implies (\forall n \in \mathbb{N}, Q'(n))$:</u>
Suppose $\forall j \in \{1, \ldots, len(A)\}, Q(j)$.

Then, fix $n \in \mathbb{N}$ and suppose $0 \leq n < len(A)$.

Because $n \in \{0, ..., len(A) - 1\}$, it follows that $(n + 1) \in \{1, \ldots, len(A)\}$.

By assumption, $Q(n + 1)$.

Thus, $\forall n \in \mathbb{N}, Q'(n)$.

<u>$(\forall j \in \{1, \ldots, len(A)\}, Q(j)) \impliedby (\forall n \in \mathbb{N}, Q'(n))$:</u>
Suppose $\forall n \in \mathbb{N}, Q'(n)$.

Let $j \in \{1, \ldots, len(A)\}$.

Then, $(j-1) \in \mathbb{N}$.

It follows that $0 \leq j-1 \leq len(A)-1$.

Since $len(A)-1 < len(A)$, $0 \leq j-1 < len(A)$.

By assumption, $Q((j-1)+1)$.

Thus, $\forall j \in \{1, \ldots, len(A)\}, Q(j)$.

<u>Conclusion:</u>
Therefore, $\forall j \in \{1, \ldots, len(A)\}, Q(j) \iff \forall j \in \mathbb{N}, Q'(n)$.

$\square$

Question #3

As follows below, Q6-Q10 respectively represent questions 6 through 10 from pp. 64-66 of the course textbook.

Q6:

Consider the following code:

```
1 def f(x):
2     """Pre: x is a natural number"""
3     a = x
4     y = 10
5     while a > 0:
6         a -= y
7         y -= 1
8     return a * y
```

(a): Loop Invariant Which Characterizes `a` and `y`:

For arbitrary natural n...

Let $i_1 = 0$ and $i_n = i_{n-1} + 1$.

Let y_n be the value of y before the $(n+1)$th iteration. By *Line 4* (initializes $y = 10$) and *Line 7* (decrements y by 1) of the program, $y_n = 10 - \sum_{q=1}^{n} 1 = 10 - n \times 1 = 10 - n$.

Denote the loop invariant:

$$P(j) : (a_j = x - \sum_{k=0}^{i_j - 1} y_k) \land (y_j = 10 - j)$$

For example, before the 1[st] iteration, $a_1 = x - \sum_{k=0}^{i_1-1} y_k = x - \sum_{k=0}^{0-1} y_k = x - 0 = x$.

Before the 2[nd] iteration, $a_2 = x - \sum_{k=0}^{i_2-1} y_k = x - \sum_{k=0}^{1-1} y_k = x - y_0 = x - 10$.

(b): Why This Function Fails to Terminate

Suppose $x > \sum_{k=1}^{10} k = 55$.

By $P(j)$, before the 11[th] iteration, $a_{11} = x - \sum_{k=0}^{i_{11}-1} y_k = x - \sum_{k=0}^{10-1} y_k = x - \sum_{k=0}^{9}(10 - k) =$

$x - [10\sum_{k=0}^{9}(1) - \sum_{k=0}^{9}(k)] = x - [10(10) - \frac{9(9+1)}{2}] = x - [100 - 45] = x - 55.$

Since $x > 55$, it follows that $a_{11} = x - 55 > 0$.

As well, y_{10} (the value of y after the 11^{th} iteration) is $10 - 11 = -1$.

Notice that in all subsequent iterations, a will decrement by $y_n < 0|_{n \geq 11}$ (where n is the iteration number of the corresponding iteration).

Since a decrements by a negative number subsequently, the loop causes a to grow large, thereby retaining $a > 0$.

Thus, the function fails to terminate for $x > 55$ (because $\neg(a > 0)$ is never satisfied).

Q7:

(a) Consider the recursive program below:

```
def exp_rec(a, b):
    if b == 0:
        return 1
    else if b mod 2 == 0:
        x = exp_rec(a, b / 2)
        return x * x
    else:
        x = exp_rec(a, (b - 1) / 2)
        return x * x * a
```

<u>Preconditions:</u>

$$(b \in \mathbb{N}) \wedge (a \neq 0)$$

<u>Postconditions:</u>

Returns a^b.

Denote the following predicate:

$$P(b) : \text{The program returns } a^b.$$

<u>Claim:</u> $\forall b \in \mathbb{N}, P(b)$

Proof.
This proof explores the Principle of Complete Induction on b.

Fix $a \neq 0$.

<u>Base Case:</u>
Let $b = 0$.

Then, by *Lines 2-3* of the program, the program returns $1 = a^0 = a^b$.

Hence, $P(0)$.

<u>Induction Hypothesis:</u>
Assume for some $k \in \mathbb{N}$ and for all $l \in [0, k] \cap \mathbb{N}$, $P(l)$.

This means the program returns a^l for every l as described.

<u>Induction Step:</u>
Proceed to show $P(k+1)$ with case analysis:

Case 1 - Suppose $(k+1)(mod\ 2) \not\equiv 0$*:*
Then, program again enters the `else` statement in *Line 7*.

Here, the program sets x to `exp_rec(a, ((k + 1) - 1) / 2)`.

Notice that `exp_rec(a, ((k + 1) - 1) / 2)` $=$ `exp_rec(a, (k / 2))`.

Since $(k+1)(\text{mod } 2) \not\equiv 0$, it must be that $k(\text{mod } 2 \equiv 0)$.

Thus, $\frac{k}{2} \in \mathbb{N}$ and $\frac{k}{2} < k$.

By the Induction Hypothesis, `exp_rec(a, k / 2)` returns $a^{\frac{k}{2}}$.

Finally, the original function call returns $x \times x \times a$, which evaluates to $a^{\frac{k}{2}} \times a^{\frac{k}{2}} \times a = a^{\frac{k}{2}+\frac{k}{2}+1} = a^{k+1}$, as needed.

Thus, $P(k+1)$ holds.

Case 2 - Suppose $(k+1)(\textit{mod } 2) \equiv 0$*:*
Then, the program reaches *Line 5* and sets x to `exp_rec(a, (k + 1) / 2)`.

Notice that $\frac{k+1}{2} \in \mathbb{N}$ and $\frac{k+1}{2} \leq k$.

By the Induction Hypothesis, `exp_rec(a, (k + 1) / 2)` returns $a^{\frac{k+1}{2}}$.

Finally, the original function call returns $x \times x$, evaluating to $a^{\frac{k+1}{2}} \times a^{\frac{k+1}{2}} = a^{\frac{k+1}{2}+\frac{k+1}{2}} = a^{k+1}$, as needed.

Thus, P(k + 1) holds.

<u>Conclusion:</u>
Therefore, $P(k+1)$ holds in all cases.

By the Principle of Complete Induction, $\forall b \in \mathbb{N}, P(b)$.

$\square$

(b) Consider the iterative version of the previous program:

```python
def exp_iter(a, b):
    ans = 1
```

```
    mult = a
    exp = b
    while exp > 0:
        if exp mod 2 == 1:
            ans *= mult
        mult = mult * mult
        exp = exp // 2
    return ans
```

Preconditions:

$$(b \in \mathbb{N}) \land (a \neq 0)$$

Postconditions:

$$\text{Returns } a^b.$$

Claim: For all natural b, the program returns a^b and terminates.

Proof.
Fix $a \neq 0$ and $b \in \mathbb{N}$.

Loop Invariant Proof:
Denote the Loop Invariant:

$$P(i) : a^b = \texttt{mult}_i^{\texttt{exp}_i} \times \texttt{ans}_i$$

To prove the loop invariant, this proof explores the Principle of Simple Induction on $i \in [1, \lfloor log_2 b \rfloor + 2] \cap \mathbb{N}$.

Base Case:
Let $i = 1$.

Then, the program retains the values $\texttt{mult}_1 = a$, $\texttt{exp}_1 = b$, `ans_1 = 1`.

Notice that $a^b = a^b \times 1 = \texttt{mult}_1^{\texttt{exp}_1} \times \texttt{ans}_1$.

Hence, at the beginning of the 1^{st} iteration, $P(1)$.

Induction Hypothesis:
Assume for some $k \in [1, \lfloor log_2 b \rfloor + 1] \cap \mathbb{N}, P(k)$;

$$P(k) : (a^b = \texttt{mult}_k^{\texttt{exp}_k} \times \texttt{ans}_k).$$

Induction Step:
Notice that $\texttt{exp}_k = \lfloor \frac{b}{2^{k-1}} \rfloor$.

Because $k - 1 \leq \lfloor log_2 b \rfloor$, then $2^{k-1} \leq 2^{\lfloor log_2 b \rfloor} \leq 2^{log_2 b} = b$.

Therefore, it follows that $\frac{b}{2^{k-1}} \geq 1$.

Thus, $\texttt{exp}_k = \lfloor \frac{b}{2^{k-1}} \rfloor \geq 1$, and the following iteration runs.

Then, the program yields the following values:

$$\begin{cases} \texttt{mult}_{k+1} = \texttt{mult}_k \times \texttt{mult}_k = \texttt{mult}_k^2, & \text{by } \textit{Line 8} \\ \texttt{exp}_{k+1} = \left\lfloor \frac{\texttt{exp}_k}{2} \right\rfloor, & \text{by } \textit{Line 9} \end{cases}$$

Notice that $\texttt{mult}_{k+1}^{\texttt{exp}_{k+1}} = (\texttt{mult}_k^2)^{\lfloor \frac{\texttt{exp}_k}{2} \rfloor} = \texttt{mult}_k^{2\lfloor \frac{\texttt{exp}_k}{2} \rfloor}$.

Proceed to show $P(k+1)$ with case analysis:

Case 1 - Suppose $exp_k (mod\ 2) \equiv 1$:
By *Lines 6-7* of the program, $\texttt{ans}_{k+1} = \texttt{ans}_k \times \texttt{mult}_k$.

So, $(\texttt{mult}_{k+1}^{\texttt{exp}_{k+1}}) \times (\texttt{ans}_{k+1}) = (\texttt{mult}_k^{2\lfloor \frac{\texttt{exp}_k}{2} \rfloor}) \times (\texttt{ans}_k \times \texttt{mult}_k)$.

Since $\texttt{exp}_k (\text{mod } 2) \equiv 1$, it follows that $2\lfloor \frac{\texttt{exp}_k}{2} \rfloor = \cancel{2}(\frac{\texttt{exp}_k - 1}{\cancel{2}}) = \texttt{exp}_k - 1$.

Thus,

$$
\begin{aligned}
(\texttt{mult}_k^{2\lfloor \frac{\texttt{exp}_k}{2} \rfloor}) \times (\texttt{ans}_k \times \texttt{mult}_k) &= (\texttt{mult}_k^{\texttt{exp}_k - 1}) \times (\texttt{mult}_k \times \texttt{ans}_k) \\
&= \texttt{mult}_k^{\texttt{exp}_k - 1} \times \texttt{mult}_k \times \texttt{ans}_k \\
&= (\texttt{mult}_k^{\texttt{exp}_k - 1} \times \texttt{mult}_k) \times \texttt{ans}_k \\
&= (\texttt{mult}_k^{(\texttt{exp}_k - 1)+1}) \times \texttt{ans}_k \\
&= \texttt{mult}_k^{\texttt{exp}_k} \times \texttt{ans}_k \\
&= a^b,
\end{aligned}
$$

by the Induction Hypothesis.
Therefore, $(\texttt{mult}_{k+1}^{\texttt{exp}_{k+1}}) \times (\texttt{ans}_{k+1}) = a^b$; $P(k+1)$ holds.

Case 2 - Suppose $exp_k(mod\ 2) \not\equiv 1$*:*
By *Line 6* of the program, *Line 7* does not run.

Hence, $\texttt{ans}_{k+1}$ retains the value as represented by $\texttt{ans}_k$; $\texttt{ans}_{k+1} = \texttt{ans}_k$.

So, $(\texttt{mult}_{k+1}^{\texttt{exp}_{k+1}}) \times (\texttt{ans}_{k+1}) = (\texttt{mult}_k^{2\lfloor \frac{\texttt{exp}_k}{2} \rfloor}) \times \texttt{ans}_k$.

Notice that $\texttt{exp}_k(\text{mod } 2) \not\equiv 1 \iff \texttt{exp}_k(\text{mod } 2) \equiv 0$.

So, $2\lfloor \frac{\texttt{exp}_k}{2} \rfloor = \cancel{2}(\frac{\texttt{exp}_k}{\cancel{2}}) = \texttt{exp}_k$.

Then, it follows that $(\texttt{mult}_k^{2\lfloor \frac{\texttt{exp}_k}{2} \rfloor}) \times \texttt{ans}_k = (\texttt{mult}_k^{\texttt{exp}_k}) \times \texttt{ans}_k = a^b$, by the Induction Hypothesis.

Still, $(\texttt{mult}_{k+1}^{\texttt{exp}_{k+1}}) \times (\texttt{ans}_{k+1}) = a^b$; $P(k+1)$ likewise holds.

Conclusion of Loop Invariant:
Collectively, $P(k+1)$ holds in all cases.

By the Principle of Simple Induction, $P(i)$ holds for all $i \in [1, \lfloor log_2 b \rfloor + 2]$.

Program Termination Proof

Notice that *Line 9* of the program performs floor division by 2 on `exp` in each iteration.

From continual division, `exp` eventually becomes small enough that it reaches 0 through the next floor division by 2.

Since the program's loop requires `exp` > 0 to run, having `exp` reach 0 indeed terminates the loop.

Conclusion:

Therefore, this program is both correct (by the loop invariant) and terminates.

$\square$

Q8

Consider the following linear time program:

```python
def majority(A):
    """
    Pre: A is a list with more than half its entries equal to x
    Post: Returns the majority element x
    """
    c = 1
    m = A[0]
    i = 1
    while i <= len(a) - 1:
        if c == 0:
            m = A[i]
            c = 1
        else if A[i] == m:
            c += 1
        else:
            c -= 1
        i += 1
    return m
```

Denote the following predicate:

$$P(n) : somethinghere$$

<u>Claim:</u> expresshowthisiscorrect

Proof.
wordsgohere

$\square$

Q9
Consider the bubblesort algorithm as follows:

```
    def bubblesort(L):
        """
        Pre: L is a list of numbers
        Post: L is sorted
        """
        k = 0
        while k < len(L):
            i = 0
            while i < len(L) - k - 1:
                if L[i] > L[i + 1]:
                    swap L[i] and L[i + 1]
                i += 1
            k += 1
```

(a): Denote the inner loop's invariant:

$$P(n) : somethinghere$$

<u>Claim:</u> proveinnerloop

Proof.

wordsgohere

$\square$

(b): Denote the outer loop's invariant:

$$P(n) : \textit{somethinghere}$$

<u>Claim:</u> proveouterloop

Proof.
wordsgohere

$\square$

(c): Denote the following predicate:

$$P(n) : \textit{somethinghere}$$

<u>Claim:</u> expresshowthisiscorrect

Proof.
wordsgohere

$\square$

Q10

Consider the following generalization of the `min` function:

```
    def extract(A, k):
        pivot = A[0]
        # Use partition from quicksort
        L, G = partition(A[1, ..., len(A) - 1], pivot)
        if len(L) == k - 1:
            return pivot
        else if len(L) >= k:
            return extract(L, k)
        else:
            return extract(G, k - len(L) - 1)
```

(a): Proof of Correctness

$$P(n) : something here$$

Claim: proofofcorrectnessclaim

Proof.
wordsgohere

□

(b): Worst-Case Runtime
wordsgohere

Question #4

As follows below, VI, VII, X, XII, and XIV respectively represent questions 6, 7, 10, 12, and 14 from pp. 46-48 of the course textbook.

VI
Let $T(n)$ be the number of binary strings of length n in which there are no consecutive 1's. So, $T(0) = 1, T(1) = 2, T(2) = 3, ...,$ etc.

(a): Recurrence for $T(n)$:
recurrencehere

(b): Closed Form Expression for $T(n)$:
closedformhere

(c): Proof of Correctness of Closed Form Expression
Denote the following predicate:

$$P(n) : \textit{somethinghere}$$

Claim: expresshowthisiscorrect

Proof.
wordsgohere

□

VII
Let $T(n)$ denote the number of distinct full binary trees with n nodes. For example, $T(1) = 1$, $T(3) = 1$, and $T(7) = 5$. Note that every full binary tree has an odd number of nodes.
Recurrence for $T(n)$:
recurrencehere

$$P(n) : \textit{somethinghere}$$

Claim: $T(n) \geq (\frac{1}{n})(2)^{(n-1)/2}$

Proof.
wordsgohere

∎

X

A *block* in a binary string is a maximal substring consisting of the same symbol. For example, the string `0100011` has four blocks: `0`, `1`, `000`, and `11`. Let $H(n)$ denote the number of binary strings of length n that have no odd length blocks of `1`'s. For example, $H(4) = 5$:

$$0000 \quad 1100 \quad 0110 \quad 0011 \quad 1111$$

Recursive Function for $H(n)$:

$$P(n) : somethinghere$$

Claim: proveouterloop

Proof.
wordsgohere

∎

Closed Form for H (Using Repeated Substitution):

XII

Consider the following function:

```
def fast_rec_mult(x, y):
n = length of x  # Assume x and y have the same length
if n == 1:
    return x * y
else:
    a = x // 10^(n // 2)
```

```
        b = x % 10^(n // 2)
        c = y // 10^(n // 2)
        d = y % 10^(n // 2)
        p = fast_rec_mult(a + b, c + d)
        r = fast_rec_mult(a, c)
        u = fast_rec_mult(b, d)

        return r * 10^n + (p - r + u) * 10^(n // 2) + u
```

Worst-Case Runtime Analysis:

wordsgohere

XIV

Recall the recurrence for the worst-case runtime of quicksort:

$$\begin{cases} c, & \text{if } n \leq 1; \\ T(|L|) + T(|G|) + dn, & \text{if } n > 1. \end{cases}$$

where L and G are the partitions of the list.

For simplicity, ignore that each list has size $\frac{n-1}{2}$.

(a): Assume the lists are always evenly split; that is, $|L| = |G| = \frac{n}{2}$ at each recursive call.

Tight Asymptotic Bound on the Runtime of Quicksort:

determinehere

(b): Assume the lists are always very unevenly split; that is, $|L| = n - 2$ and $|G| = 1$ at each recursive call.

Tight Asymptotic Bound on the Runtime of Quicksort:

determinehere