8 Sept. 2021

What are the Fib. #s?

$f_1 = f_2 = 1$

$f_n = f_{n-1} + f_{n-2} \quad \forall n > 2,\ n \in \mathbb{Z}$

<u>Fun Fact</u>: $\boxed{\forall k \in \mathbb{Z}_+,\ f_{3k} \text{ is even!}}$

<u>Proof</u>: (use induction!)

Consider $k=1$. Then, $f_{3k} = f_{3 \cdot 1} = f_3$

$= f_2 + f_1$

$= 1 + 1$

$= 2$, which is an even #.

(we've completed our base case).

Let $x \in \mathbb{Z}$ s.t. $x \geq 1$.
Assume that f_{3x} is even. } ind. assumption

Now, we look at $f_{3(x+1)}$

$$f_{3(x+1)} = f_{3x+3} \quad \text{by factoring}$$
$$= f_{3x+2} + f_{3x+1} \quad \text{by definition}$$
$$= (f_{3x+1} + f_{3x}) + f_{3x+1} \quad \text{by def'n of } f_{3x+2}$$

Since f_{3x} is even, we know that there exists an integer y such that $f_{3x} = 2y$

Therefore, $f_{3(x+1)} = 2f_{3x+1} + 2y$

$= 2(f_{3x+1} + y)$.

Since f_{3x+1} and y are integers, $f_{3(x+1)}$ is even! ∎

Recursion Invariants

CSCI 432, Fall 2021

September 8, 2021

Recursion Invariants: Proving Partial Correctness

"If it terminates, then it is correct."
How? I, M, E

Recall theHANOI algorithm (given in Algorithm 1); see [1, Ch. 1].

Algorithm 1 HANOI($n, src, dest, tmp$)

Input: $n \in \mathbb{N}$, and three towers with disks: $src, dest, tmp$ such that P
Output: R (see below)

```
if n > 0 then
    HANOI(n − 1, src, tmp, dest)      ← Assert R(n−1, src, tmp, dest)
    move top disk from src to dest
    HANOI(n − 1, tmp, dest, src)      ← Assert R(n−1, tmp, dest, src)
end if
```

Assert $R(n,$ ~~tmp~~ $src, dest, tmp)$
Return

1. Suppose we have N disks total. What are the assumptions on the input to the initial call HANOI($N, src, dest, tmp$)?

   Answer:

2. What does it mean for HANOI($N, src, dest, tmp$) to execute correctly? What does it return / what does it need to accomplish?

   Answer:

   Going forward, we call this statement Q.

3. For a general call to the recursive algorithm what are the assumptions on the input (For convenience, suppose the call is: $\text{HANOI}(n, A, B, C)$).

Answer:

Going forward, we will call these assumptions P.

4. What is the recursion invariant?

The recursion invariant is the (compound) statement R that can fill in the blank of the following sentence: *At each recursive call* $\text{HANOI}(n, A, B, C)$*, R is satisfied (i.e., true).* Moreover, R is such that we can use it to prove INITIALIZATIOn, MAINTENCE, and END.

For HANOI, the recursion invariant R is:

- There are currently no violations of smaller disks on larger disks. (Note: the world would crumble if this were violated at any time), AND
- the n smallest disks are now on B. (Note: they began on A), AND
- all other disks are where they were when first called. ~~no other disks have moved.~~

$R = R(n, A, B, C)$

5. INITIALIZTION This is like the base case for recursion. Colloquially, we ask "Why is this true for the smallest input?" (And, what is those inputs that would allow us to return without a recursive call?) More formally, we can say: If $n = n_0, A, B, C$ satisfy P, then after the call to $\text{HANOI}(n_0, A, B, C)$, the recursion invariant R is satisfied.

Answer:

Note: sometimes, just as in induction, there may be more than one base case.

6. MAINTENCE: This part is JUST like induction. Let $k \in \mathbb{N}$ such that $k \geq n_0$. Assume that, for all $n \in \mathbb{N}$ such that $n_0 \leq n \leq k$, the recursion invariant R holds after a call to $\text{HANOI}(n, *, *, *)$. (That was the equivalent to the inductive assumption). Now, we must prove that R holds after $\text{HANOI}(k+1, A, B, C)$. (Hint: use the line numbers as you walk through the algorithm).

   Answer:

7. END: This is where we diverge from induction. Since algorithms are finite, we can't go on forever. Colloquially, we say "if the initial call $\text{HANOI}(N, src, dest, tmp)$ finishes executing, then all N disks (which were initially on src) have moved to dst." More formally, we can phrase this as: If the recursion invariant holds and the algorithm completes execution, then the post-condition Q is satisfied.

   Answer: and the recurs. Inv. holds

References

[1] ERICKSON, J. *Algorithms.* Independently published open access, June 2019.

Things to try:

1. Walk through / step through a complete execution for N = 2 or N = 3.

Notes:

1. Is 0 a natural #? There is debate!

   In this class, I (try to) use the convention

   $\mathbb{N} := \{0, 1, 2, 3, \dots\}$

   $\mathbb{Z}_+ := \{1, 2, 3, \dots\}$

2. A full proof of correctness shows:

   ① That the algorithm terminates. } Runtime (Dec. fcn.)

   ② If the algorithm terminates, then it is correct. } Partial correctness / LI / RI

   $\therefore$ by Modus Ponens, The algorithm is correct.