Class on
11 Sept 2023
day 2 of worksheet

Recursion Invariants

CSCI 432, Fall 2021

September 8, 2023

Recursion Invariants: Proving Partial Correctness

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

1

2 Quicksort

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)
    move top disk from src to dest
    HANOI(n − 1, tmp, dest, src)
end if
```

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

   Going forward, we will call these assumptions P.

2. What does it mean for $\textsc{Hanoi}(n, src, dest, tmp)$ 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: $\textsc{Hanoi}(k, A, B, C)$).

   Answer:

   Note: when making a recursive call, we must justify that we have met these conditions.

4. What is the recursion invariant?

   The recursion invariant is statement (i.e., a sentence that can be evaluated to TRUE/FALSE). In particular, R is (in general) the statement, *Each recursive call* $\textsc{Hanoi}(k, A, B, C)$ *executes correctly.* What does that mean in our case? Well, it means

   - 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 k smallest disks are now on B, AND
   - no other disks besides the k smallest have moved (since right before this call).

   We use the recursion invariant to prove INITIALIZATION, MAINTENANCE, and END. So, sometimes we may see the invariant right away. Other times, we may need to try the proofs then revisit the invariant (as we might realize that we forgot something).

5. INITIALIZATION This is like the base case of induction. Colloquially, we ask "Why is this true for the smallest input?" (And, what are those inputs that would allow us to return without a recursive call?) More formally, we can say: If $n_0 = 0$, then after the call to HANOI(n_0, A, B, C), the recursion invariant R is satisfied.

Answer:

< cases where no recursion happens >

< might itself be a recursive call >

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

6. MAINTENANCE: This part is JUST like the inductive step of induction. Let $k \in \mathbb{N}$ such that $k \geq n_0$. Assume that, for all $k' \in \mathbb{N}$ such that $n_0 \leq k' \leq k$, the recursion invariant R holds after a call to HANOI($k', *, *, *$). (That was the equivalent to the inductive assumption). Now, we must prove that R holds after HANOI($k+1, A, B, C$). (Hint: for this, we will almost always need to use the line numbers as we walk through the algorithm to explain how the maintenance step works).

Answer:

Since HANOI(k+1, A, B, C) was called, we assume the inputs are correct/follow the assumptions given in #3, which means k+1 smallest discs are on A.
Since the recursion fairy handled Hanoi(n-1, src, tmp, dest) on line 2, I know (RI goes here): ① there are no current violations ② the $n-1$ $^{k+1-1=k}$ smallests disks are now on peg B, and ③ no other discs have moved.
So now, the ~~k+1~~ k smallest are on B, the (k+1)-st smallest is on A & no current violations.

(Reset/ explain what's going on)

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 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, if the preconditions P are satisfied, and if the algorithm completes execution, then the post-condition Q is satisfied. (For shorthand, I might write $R \wedge P \wedge T \implies Q$).

Answer:

References

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

input: <usu. assumptions>

output: <explains <u>what</u> it is supposed to do>

MyAlgorithm (a, b, c)

$P_1 \wedge$ LINE 1 $\Rightarrow Q_1$

$P_2 \wedge$ LINE 2 $\Rightarrow Q_2$

$P_3 \wedge$ LINE 3 $\Rightarrow Q_3$

$P_4 \wedge$ LINE 4 $\Rightarrow Q_4$

$\vdots$

$P_N \wedge$ LINE N $\Rightarrow Q_N$

P_{N+1} Return x

"preconditions" "post conditions"

P_1 = assumptions on my input

Really (for the most part) $Q_1 = P_2$

↳ other than "pointr" to line of assembly code prove this is correct in HW-2

```
QUICKSORT(A[1..n]):
  if (n > 1)
        Choose a pivot element A[p]
        r ← PARTITION(A, p)
        QUICKSORT(A[1..r − 1])      ⟨⟨Recurse!⟩⟩
        QUICKSORT(A[r + 1..n])      ⟨⟨Recurse!⟩⟩
```

```
PARTITION(A[1..n], p):
  swap A[p] ↔ A[n]
  ℓ ← 0                  ⟨⟨#items < pivot⟩⟩
  for i ← 1 to n − 1
      if A[i] < A[n]
          ℓ ← ℓ + 1
          swap A[ℓ] ↔ A[i]
  swap A[n] ↔ A[ℓ + 1]
  return ℓ + 1
```

Figure 1.8. Quicksort