CSCI 432 Handout 05: Recursion Invariants

Name: _____

Collaborators: _____

3 February 2026

Recursion Invariants: Proving Partial Correctness

Recall theHANOI algorithm (given in Algorithm 1); see [1, Ch. 1]. In this problem, have three towers, each with a set of discs on them. The stacking rule is that all discs are a unique size, all disks are stacked on towers, and no smaller disc is underneath a larger one.

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

Input: $n \in \mathbb{N}$, and three towers with disks: $src, dest, tmp$ such that [[See Q1 below]].
Output: [[See Q2 below]].

```
1: if n > 0 then
2:     HANOI(n − 1, src, tmp, dest)
3:     move top disk from src to dest
4:     HANOI(n − 1, tmp, dest, src)
5: 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 (for the preconditions).

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 (for postcondition; the letter P was already taken).

3. For a general call to the recursive algorithm what are the assumptions on the input (For convenience, suppose the call is: 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 R is statement (i.e., a sentence that can be evaluated to TRUE/FALSE). In particular, R is (in general) the statement, *Each recursive call* HANOI(k, A, B, C) *executes correctly and gets us closer to the end result.* 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 we made 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:

   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:

7. END: This is where we diverge from induction. Because 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.