Problem set 1

Theodore S. Norvell

6892

Problem 0: Insertion sort. Draw a picture of the invariant for (the outermost loop of) insertion sort. Write it out mathematically. (It may help to define some predicates.)

Problem 1: Heap sort. Do the same for heap sort. Heap sort has two phases. Each phase will need an invariant.

Problem 2: String search. Suppose we want to search a string t for the first occurrence of a pattern p. Let $M = t.\,\text{length}$ and $N = p.\,\text{length}$. We want to find the minimum k such that either

$$p = t[k, ..k + N]$$

or

$$k + N > M$$

(a) Think about doing this with one loop (no nesting). Draw a picture of the invariant.

(b) Write the invariant out mathematically.

(c) Write the algorithm out.

From here on it gets interesting. I'll assume your solution in part (c) is based on sliding p one place to the right whenever there is a failure to match.

(d) Think about whether there is a way to use the information that there is a partial march to instead slide the pattern by a (possibly) larger amount without breaking the invariant. The amount to slide should be a function of p and the location in p where the failure happens. Rewrite the algorithm to use this function. Avoid checking items of t that have already matched. What is that function?

(e) Assume that the items t come from a small set A. Is there a way to slide by a (possibly) even larger amount that is a function of p, the location of the failure, and the value of the item of t where the failure occurs. The new algorithm should not need to read any item of t more than once.

(f) Write an algorithm to precompute the function you developed in (e), based on p. I.e. to make, given p, a table of slide values indexed by the location of the failure and the item where the failure occurred.

(g) What is the time complexity of your algorithm?

Problem 3. Linked list insertion.

Suppose. *head* points to the first node[0] of a nonempty, finite, sorted linked list. We wish to

[0] You may assume that each node is an object of a class

```
class Node
    var data : int
    var next : Node
end Node
```

insert an integer i into the list so that it remains sorted. You will need a loop. (a) Draw a picture of the invariant for the loop.

(b) Formalize the assumptions that the linked list is nonempty, finite, and sorted. To do this, assume we have a finite set of pointers P and functions $data : P \to \mathbb{Z}$ and $next : P \to P \cup \{null\}$ where $null$ is a value not in P. It may help to define a family of partial functions $next^i$ as follows

$$\begin{aligned} next^0(p) &= p \\ next^{i+1}(p) &= next^i(next(p)), \text{ where } i \in \mathbb{N} \text{ and } next(p) \in P \end{aligned}$$

(c) Write out the invariant formally using the ideas of part (b).

(d) Complete the algorithm based on your invariant.

(e) (Deeper exploration.) Your algorithm will need to allocate a new node. How could we formally model a statement like $q :=$ **new** *Node*. I.e. what rule could we add to P.O.L. for commands like this. (Note that since allocation has a side effect as well as a value, we can't use our usual assignment rule.)

Problem 4: Fibonacci. Define fib by

$$\begin{aligned} \text{fib}(-1) &= 0 \\ \text{fib}(0) &= 1 \\ \text{fib}(i+1) &= \text{fib}(i) + \text{fib}(i-1), \text{ for all } i \in \mathbb{Z} \end{aligned}$$

Suppose we want to compute fib(n), for natural n.

$\{n \geq 0\}$
?
$\{a = \text{fib}(n)\}$

A rather unsatisfactory solution to this problem is following algorithm.

$\{n \geq 0\}$
$i := 0$
$a := 1$
$\{0 \leq i \leq n \wedge a = \text{fib}(i)\}$
// Variant $n - i$
while $i < n$ do
    $a := a + \text{fib}(i-1)$
    $i := i + 1$
end while
$\{a = \text{fib}(n)\}$

This algorithm is unsatisfactory as it presupposes that we can calculate the fib function for arbitrary integers; and if that were the case, it would be simpler just to write $a := \text{fib}(n)$.

(a) Use proof outline logic to verify this algorithm

(b) Introduce a tracking variable to eliminate the need to compute fib($i-1$) in the loop.

Problem 5: Binary search revisited.

(a) Consider the problem of finding an up edge of a function from the notes

$\{ \neg A(m) \wedge A(n) \wedge m < n \}$
?

$\{\ \neg A(p) \wedge A(p+1)\ \}$

Find a data refinement of this problem so that it transforms into the following problem

$\{\ x.length = N\ \}$
?
$\{\ -1 \leq p < N \wedge (p = -1 \vee x(p) \leq t) \wedge (p = N-1 \vee x(p+1) > t)\ \}$

I.e. find a suitable linking invariant that links the space A, m, n, p to the space x, t, N, p.

(b) Transform the abstract binary search algorithm to a binary search algorithm to search the sequence x.

(c) How does the invariant of your algorithm for part (b) compare to the invariant of the algorithm presented in class? For the algorithm developed in class did we make use of any additional assumptions about array x that we did not make in parts (a) and (b)?

(d) Suppose that what we really want to do is determine whether t is in x. Assume that x is nondecreasing and solve the following programming problem.

$\{\ -1 \leq p < N \wedge (p = -1 \vee x(p) \leq t) \wedge (p = N-1 \vee x(p+1) > t)\ \}$
?
$\{\ f = (t \in x\,\{0, ..N\})\ \}$