COMP 280 : Exam 1 - Sample Solutions

1. (a) Assume that $\forall x : (A(x) \rightarrow B(x))$. Either $\forall x : A(x)$, or $\exists x : \neg A(x)$. If $\forall x : A(x)$, then, since $\forall x : (A(x) \rightarrow B(x))$, we also have $\forall x : B(x)$, so $\forall x : (A(x) \rightarrow \forall x : B(x))$, so $\exists x : (A(x) \rightarrow \forall x : B(x))$. If $\exists x : \neg A(x)$, then, since $A(x)$ is false, $A(x) \rightarrow \forall x : B(x)$, so $\exists x : (A(x) \rightarrow \forall x : B(x))$, again.

   (b) Let x be a natural number n, $A(n)$ be $n \neq 0$, and $B(n)$ be $0 = 1$. Then the formula becomes:

$$\exists n : ((n \neq 0) \rightarrow \forall m : (0 = 1)) \rightarrow \forall n : ((n \neq 0) \rightarrow (0 = 1))$$

   The left hand side is true (if n is 0, then $(n \neq 0)$ is false, so $(n \neq 0) \rightarrow \forall m : (0 = 1))$, but the right hand side is false (if n is 1, then $(n \neq 0)$ is true, but it does not mean that $0 = 1$). Note that in the original formula, the fourth x binds to the nearest enclosing quantifier, so it can be renamed m in the formula above.

2. (a)
   - empty_heap?(emptyHeap) = true
   - empty_heap?(insert(heap num)) = false
   - insert(heap_extract_max(heap), heap_max(heap)) = heap if heap $\neq$ emptyHeap.
   - insert(insert(heap num1) num2) = insert(insert(heap num2) num1)
   - heap_max(emptyHeap) = error
   - heap_max(insert(heap num)) = num if num $\geq$ heap_max(heap) or heap = emptyHeap, heap_max(heap) otherwise.
   - heap_extract_max(emptyHeap) = error
   - heap_extract_max(insert(heap num)) = heap if num $\geq$ heap_max(heap) or heap = emptyHeap, insert(heap_extract_max(heap) num) otherwise.
   - heap_size(emptyHeap) = 0
   - heap_size(insert(heap num)) = 1 + heap_size(heap)
   - heap_depth(heap) = $\lceil$lg(heap_size(heap)+1)$\rceil$

   (b) Base case: $T(1) = 1 \leq 1$

   Induction: Assume $\forall k < n : T(k) \leq k$. Then: $T(n) = T(n/2) + 1 \leq n/2 + 1$ (by the induction hypothesis) $\leq n$ (for $n \geq 2$).

   Note: we use here strong induction, otherwise we could not say anything about the value of $T(n/2)$. Assuming $T(k) \leq k$ just for $k = n - 1$ is not enough in this case. Another solution was to separate the odd / even cases, and prove that $T(n) \leq n$ implied both $T(2n) \leq 2n$ and $T(2n + 1) \leq 2n + 1$ (proving it just for $T(2n)$ was proving the property only for powers of 2).

3. (a) A configuration can be represented using a tuple $\langle M_L, C_L, M_B, C_B, B\rangle$, where M_L, M_B, C_L and C_B are the number of missionaries and cannibals on the left of the river and in the boat, respectively, and $B \in \{$left, right$\}$ represents the location of the boat.

   Note: several representations were possible, and the answers to the next questions depended on the answer to this one.

   (b) If there is k missionaries in the boat, there there is $m - k$ missionaries left to place on the 2 banks, which gives $C((m - k) + 2 - 1, m - k) = C(m - k + 1, m - k) = (m - k + 1)$ possibilities (unlabeled balls and urns problem). Since k can vary between 0 and p, the total number of ways M_p to distribute the missionaries among the two sides of the river and the boat is:

$$M_p = \sum_{k=0}^{p}(m-k+1) = \sum_{k=0}^{p} m - \sum_{k=0}^{p} k + \sum_{k=0}^{p} 1 = (p+1)m - (p+1)\frac{p}{2} + (p+1) = (p+1)(m+1-\frac{p}{2})$$

   (c) By symmetry with the previous problem, there is $(c - k + 1)$ ways to distribute the cannibals among the two sides of the river and the boat, when there is k cannibals in the boat.

If the boat has two seats, then the boat can hold zero, one, or two cannibals. If the boat carries no cannibal, it can carry zero, one or two missionaries and there is $(c-0+1)((m-0+1)+(m-1+1)+(m-2+1))$ possibilities. If the boat carries one cannibal, it can carry zero or one missionaries, and there is $(c-1+1)((m-0+1)+(m-1+1))$ possibilities. If the boat carries two cannibals, then it does not carry missionaries, and there is $(c-2+1)(m-0+1)$ possibilities. So the total is:

$$(c+1)(m+1)+(c+1)m+c(m+1)+(c+1)(m-1)+cm+(c-1)(m+1) = 6cm+2m+2c-1$$

(d) Let S be the set of all safe configurations that can be reached from the starting configuration. Then we can define S inductively as:

- $\langle m, c, 0, 0, \text{left}\rangle \in S$ (initial state)
- If $\langle M_L, C_L, M_B, C_B, \text{left}\rangle \in S$ and $C_L > 0$ and $M_B + C_B < p$ and ($M_B > C_B$ or $M_B = 0$) then $\langle M_L, C_L - 1, M_B, C_B + 1, \text{left}\rangle \in S$ (one cannibal on the left side of the river boarded the boat)
- If $\langle M_L, C_L, M_B, C_B, \text{left}\rangle \in S$ and $C_B > 0$ and ($M_L > C_L$ or $M_L = 0$) then $\langle M_L, C_L + 1, M_B, C_B - 1, \text{left}\rangle \in S$ (one cannibal left the boat for the left side of the river)
- If $\langle M_L, C_L, M_B, C_B, \text{left}\rangle \in S$ and $M_L > 0$ and $M_B + C_B < p$ and ($M_L > C_L$ or $M_L = 1$) then $\langle M_L - 1, C_L, M_B + 1, C_B, \text{left}\rangle \in S$ (one missionary on the left side of the river boarded the boat)
- If $\langle M_L, C_L, M_B, C_B, \text{left}\rangle \in S$ and $M_B > 0$ and ($M_B > C_B$ or $M_B = 1$) then $\langle M_L + 1, C_L, M_B - 1, C_B, \text{left}\rangle \in S$ (one missionary left the boat for the left side of the river)
- If $\langle M_L, C_L, M_B, C_B, \text{left}\rangle \in S$ then If $\langle M_L, C_L, M_B, C_B, \text{right}\rangle \in S$ (boat crossing the river from left to right)
- If $\langle M_L, C_L, M_B, C_B, \text{right}\rangle \in S$ then If $\langle M_L, C_L, M_B, C_B, \text{left}\rangle \in S$ (boat crossing the river from right to left)
- If $\langle M_L, C_L, M_B, C_B, \text{right}\rangle \in S$ and $c - C_L - C_B > 0$ and $M_B + C_B < p$ and ($M_B > C_B$ or $M_B = 0$) then $\langle M_L, C_L, M_B, C_B + 1, \text{right}\rangle \in S$ (one cannibal on the right side of the river boarded the boat)
- If $\langle M_L, C_L, M_B, C_B, \text{right}\rangle \in S$ and $C_B > 0$ and ($m - M_L - M_B > c - C_L - C_B$ or $m - M_L - M_B = 0$) then $\langle M_L, C_L, M_B, C_B - 1, \text{right}\rangle \in S$ (one cannibal left the boat for the right side of the river)
- If $\langle M_L, C_L, M_B, C_B, \text{right}\rangle \in S$ and $m - M_L - M_B > 0$ and $M_B + C_B < p$ and ($m - M_L - M_B > c - C_L - C_B$ or $m - M_L - M_B = 1$) then $\langle M_L, C_L, M_B + 1, C_B, \text{right}\rangle \in S$ (one missionary on the right side of the river boarded the boat)
- If $\langle M_L, C_L, M_B, C_B, \text{right}\rangle \in S$ and $M_B > 0$ and ($M_B > C_B$ or $M_B = 1$) then $\langle M_L, C_L, M_B - 1, C_B, \text{right}\rangle \in S$ (one missionary left the boat for the right side of the river)

(e) Everyone can safely cross the river if $\langle 0, 0, 0, 0, \text{right}\rangle \in S$.

4. We are first going to prove that (*insert* E L) contains both E and the elements of L, by induction on L.

Base case. If L is *empty*, then:

```
(insert E empty)
=
(cond [(empty? empty) (list E)]
      [else (if (> E (first empty))
                (cons (first empty) (insert E (rest empty)))
                (cons E empty))])
=
(cond [true (list E)]
      [else (if (> E (first empty))
                (cons (first empty) (insert E (rest empty)))
                (cons E empty))])
=
(list E)
```

so E is in the result of (*insert* E L), as well as all the (non-existent) elements of L.

Induction step. Assume (*insert* E L) contains E and all the elements of L. Let's prove that (*insert* E (*cons* x L)) contains E and all the elements of (*cons* x L).

```
(insert E (cons x L))
=
(cond [(empty? (cons x L)) (list E)]
      [else (if (> E (first (cons x L)))
                (cons (first (cons x L)) (insert E (rest (cons x L))))
                (cons E (cons x L)))])
=
(cond [false (list E)]
      [else (if (> E (first (cons x L)))
                (cons (first (cons x L)) (insert E (rest (cons x L))))
                (cons E (cons x L)))])
=
(if (> E (first (cons x L)))
  (cons (first (cons x L)) (insert E (rest (cons x L))))
  (cons E (cons x L)))
=
(if (> E x)
  (cons (first (cons x L)) (insert E (rest (cons x L))))
  (cons E (cons x L)))
```

Then, either E is bigger than x, or it is not.

If it is:

```
(if (> E x)
  (cons (first (cons x L)) (insert E (rest (cons x L))))
  (cons E (cons x L)))
=
(if true
  (cons (first (cons x L)) (insert E (rest (cons x L))))
  (cons E (cons x L)))
=
(cons (first (cons x L)) (insert E (rest (cons x L))))
=
(cons x (insert E L))
```

But by the induction hypothesis, (*insert* E L) contains E and all the elements of L, so (*cons* x (*insert* E L)) contains E, and all the elements of (*cons* x L).

If E is less then or equal to x, then:

```
(if (> E x)
   (cons (first (cons x L)) (insert E (rest (cons x L))))
   (cons E (cons x L)))
=
(if false
   (cons (first (cons x L)) (insert E (rest (cons x L))))
   (cons E (cons x L)))
=
(cons E (cons x L))
```

which again contains E and all the elements of (*cons* x L), which completes the proof.

Now that we know that *insert* works as expected, we are going to show that *isort* works as expected, by proving by induction on L that (*isort* L L') contains all the elements of L and L'.

Base case. If L is *empty*, then:

```
(isort empty L')
=
(cond [(empty? empty) L']
      [else (isort (rest empty)
                   (insert (first empty) L'))])
=
(cond [true L']
      [else (isort (rest empty)
                   (insert (first empty) L'))])
=
L'
```

which contains all the (non-existent) elements of L and all the elements of L'.

Induction step. Assume (*isort* L L') contains all the elements of L and L'. Let's prove that (*isort* (*cons* x L) L') contains all the elements of (*cons* x L) and L'.

```
(isort (cons x L) L')
=
(cond [(empty? (cons x L)) L']
      [else (isort (rest (cons x L))
                   (insert (first (cons x L)) L'))])
=
(cond [false L']
      [else (isort (rest (cons x L))
                   (insert (first (cons x L)) L'))])
=
(isort (rest (cons x L))
      (insert (first (cons x L)) L'))
=
(isort L (insert x L'))
```

But we have proven before that (*insert* E L) contains E and all the elements of L, so (*insert* x L') is a list containing x and all the elements of L'.

But then, by the induction hypothesis (which applies for all accumulators, since no assumption was made about L' in the hypothesis, and since the induction is on L), (*isort* L (*insert* x L')) contains all the elements of L and all the elements of (*insert* x L'). So (*isort* L (*insert* x L')) contains all the elements of L, x, and all the elements of L', which completes the correctness proof for *isort*.

Note: if our induction hypothesis had assumed that the accumulator was *empty*, then we would have been stuck at this point, because we would have been unable to say anything regarding (*isort* L (*insert* x L')) (since (*insert* x L') is different from *empty*). Hence the necessity of making no assumptions at all about the accumulator in our induction hypothesis (as well as in our base case).

To complete the proof of the required property, we have, by definition, (*insert-sort* L) = (*isort* L *empty*), so, given what we just proved, (*insert-sort* L) contains all the elements of L and all the (non-existent) elements of *empty*. So it contains all the elements of L.