CSE505 HW01 Sample Solution

1. Caml warm-up: See `queues.ml`.

2. Interpreter warm-up:

   (a) A heap is implemented as a set of "linked" functions. Each function stores one element of the heap and accesses the rest of the heap by calling the old heap. Newer bindings overwrite older bindings because they are "further-out" than the older bindings. Alternately, the implementation can be seen as implementing a linked list with functions.

   (b) See `interp.ml`.

   (c) The new code returns the old heap instead of the (potentially) updated heap. The affect of any assignments in s1 will be discarded. For example,

   $$ans := 1; ans := ans + 1$$

   returns 2 in the original code but 1 in alternate code.

3. Non-preemptive threads

   (a) Small-step operational semantics for IMP.

   $\boxed{H\ ;\ s\ ;\ q \to H\ ;\ s\ ;\ q\ ;\ b}$

   SKIP
   $$\frac{}{H\ ;\ \mathsf{skip}\ ;\ s{::}q \to H\ ;\ s\ ;\ q\ ;\ \mathsf{false}}$$

   ASSIGN
   $$\frac{H\ ;\ e\ \Downarrow\ c}{H\ ;\ x := e\ ;\ q \to H, x \mapsto c\ ;\ \mathsf{skip}\ ;\ q\ ;\ \mathsf{false}}$$

   SEQ1
   $$\frac{}{H\ ;\ \mathsf{skip}; s\ ;\ q \to H\ ;\ s\ ;\ q\ ;\ \mathsf{false}}$$

   SEQ2
   $$\frac{H\ ;\ s_1\ ;\ q \to H'\ ;\ s_1'\ ;\ q'\ ;\ \mathsf{false}}{H\ ;\ s_1; s_2\ ;\ q \to H'\ ;\ s_1'; s_2\ ;\ q'\ ;\ \mathsf{false}}$$

   SEQ3
   $$\frac{H\ ;\ s_1\ ;\ q \to H'\ ;\ s_0\ ;\ q'{::}s_1'\ ;\ \mathsf{true}}{H\ ;\ s_1; s_2\ ;\ q \to H'\ ;\ s_0\ ;\ q'{::}(s_1'; s_2)\ ;\ \mathsf{true}}$$

   IF1
   $$\frac{H\ ;\ e\ \Downarrow\ c \qquad c > 0}{H\ ;\ \mathsf{if}\ e\ s_1\ s_2\ ;\ q \to H\ ;\ s_1\ ;\ q\ ;\ \mathsf{false}}$$

   IF2
   $$\frac{H\ ;\ e\ \Downarrow\ c \qquad c \le 0}{H\ ;\ \mathsf{if}\ e\ s_1\ s_2\ ;\ q \to H\ ;\ s_2\ ;\ q\ ;\ \mathsf{false}}$$

   WHILE
   $$\frac{}{H\ ;\ \mathsf{while}\ e\ s\ ;\ H \to q\ ;\ \mathsf{if}\ e\ (s; \mathsf{while}\ e\ s)\ \mathsf{skip}\ ;\ q\ ;\ \mathsf{false}}$$

   SPAWN
   $$\frac{}{H\ ;\ \mathsf{spawn}(s)\ ;\ q \to H\ ;\ \mathsf{skip}\ ;\ q{::}s\ ;\ \mathsf{false}}$$

   YIELD1
   $$\frac{}{H\ ;\ \mathsf{yield}\ ;\ \cdot \to H\ ;\ \mathsf{skip}\ ;\ \cdot\ ;\ \mathsf{false}}$$

   YIELD2
   $$\frac{}{H\ ;\ \mathsf{yield}\ ;\ s{::}q \to H\ ;\ s\ ;\ q{::}\mathsf{skip}\ ;\ \mathsf{true}}$$

   (b) See `interp.ml`.

   (c) See `test1`.

4. Language properties.

(a) If $H; s; q \to H'; s; q'; b$, then q' and $'q$ have the same length.

Proof. This statement is false. Consider the rule SPAWN. In this rule q' is longer that q by one. □

(b) If $H; s; q \to H'; s'; q'; b$, then q' and q' have lengths that differ by at most one.

Proof. This is true. Proof by induction on the height of the derivation tree for the step $H; s; q \to H'; s'; q'; b$. Expression evaluation has no effect on the queue, so we can treat rules whose premises only involve expression evaluation as having height one.

Suppose that the derivation tree has height one. Then the rule applied must have been SKIP, ASSIGN, SEQ1, IF1, IF2, WHILE, SPAWN, YIELD1, or YIELD2. The first six rules and the YIELD1 rule leave the queue unchanged, and the theorem holds trivially. SPAWN adds one element to the q so the lengths are within 1. YIELD2 adds one element to the queue and removes one element, so the length of the queue before the step is the same after.

The base case holds. Now we show that it holds for $n > 1$. Assume the theorem holds for derivation trees of height $n - 1$. If the step has a derivation of height greater than 1, the bottom most rule in the derivation must be SEQ2 or SEQ3. If it is SEQ2 we know from the induction hypothesis that the length of q' differs from the length of q by at most 1. Because q and q' are the queues in the used in left and right-hand sides the conclusion, this case is complete. If the rule is SEQ3 we know from the induction hypothesis that the length of $q' :: s'_1$ is within 1 of the length of q. $q' :: (s'_1; s_2)$ has the same length as $q' :: s'_1$, and this case is complete.

Having shown that the theorem holds in all cases, we are done. □

(c) If $H; s; q$, then either s is skip and q is $\cdot$ or there exists H', s', q', and b such that $H; s; q \to H'; s'; q'; b$.

Proof. This is true. We will show that if $H; s; q$, then either s is skip and q is $\cdot$ or there exists H', s', q', and b such $H; s; q \to H'; s'; q'; b$. Furthermore, if b is true then $q' \neq \cdot$.

We prove the theorem by induction on the height of the syntax tree for s. Suppose s has height 0. Then s is skip, yield, or x := e.

If s is skip and $q = \cdot$, we are done. Otherwise $q = s :: q$", and the rule SKIP applies and b is false so we are done.

If s is yield and $q = \cdot$ the rule YIELD1 applies and b is false so we are done. Otherwise $q = s :: q$", and the rule YIELD2 applies. b is true, but skip was added to the queue, making it non-empty, so we are done.

If s is x := e the rule ASSIGN applies regardless of the state of q and b is false and we are done.

Now suppose s has height greater than 1. Then s is one of the other statement forms.

If s is if e s_1 s_2 then the rule IF1 or IF2 applies (since expressions always evaluate to constants), regardless of the state of the queue. In either case, b is false, so we are done.

If s is while e s_1 or spawn(s_1) similar reasoning applies.

Finally, suppose s is s_1;s_2. s_1 has a syntax tree of height less than that of s_1;s_2 so $H; s_1; q \to H'; s'_1; q'; b$. Suppose b is false, then SEQ2 applies regardless of the state of the queue, furthermore, b is false in the conclusion, so we are done. Now suppose b is true. We know by the induction hypothesis that the queue is not empty. Therefore, SEQ3 applies. Furthermore, since the queue in the conclusion is the same length as the queue in the premise, we still have the property that when b is true, the queue is non-empty.

Having shown the theorem holds for all s, we can conclude that the theorem is true. □

5. Static analysis

   (a) See `interp.ml`.

   (b) See `interp.ml`.

   (c) True or false

   - False: example *while* 1 (*skip*)
   - False: example *skip*
   - True: The general idea for why this is true is that `spawn_count2` determines the maximum number of spawns.
   - False: example *skip*