CSE505 HW02 Sample Solution

1. Semantics for regular expressions.

   (a) (a) $\boxed{s \in r}$

$$\text{CHAR}\ \frac{}{x \in x} \qquad \text{CONCAT}\ \frac{s_1 \in r_1 \qquad s_2 \in r_2}{s_1 s_2 \in r_1 \cdot r_2} \qquad \text{UNION-1}\ \frac{s \in r_1}{s \in r_1 \vee r_2} \qquad \text{UNION-2}\ \frac{s \in r_2}{s \in r_1 \vee r_2}$$

$$\text{STAR-1}\ \frac{}{\varepsilon \in r_1^*} \qquad \text{STAR-2}\ \frac{s_1 \in r_1 \qquad s_2 \in r_1^*}{s_1 s_2 \in r_1^*}$$

   (b) See `interp.ml`.

   (c) $(((a)^*)^*, a)$: a should be accepted but the interpreter loops infinitely. $(((a)^*)^*, b)$: b should be rejected but the interpreter loops infinitely.

   (d) Replace the rules STAR-2 in (a) with

$$\text{STAR-2}\ \frac{s_1 \in r_1 \qquad s_2 \in r_1^* \qquad s_1 \neq \varepsilon}{s_1 s_2 \in r_1^*}$$

   (e) *Proof.* Let $s \in_a r$ mean that there is a derivation of $s \in r$ using part (a) semantics. RULE_a is rule RULE from part (a) semantics. Define $s \in_e r$ and RULE_e similarly. Suppose $s \in_a r$. We prove that there is a derivation $s \in_e r$ by induction on the height of the derivation tree.

   Base cases: Suppose $s \in_a r$ can be derived in one step. The the rule at the bottom must be CHAR_a or STAR-1_a. In the first case, we have that $r = x$ for some x such that $x = s$. Then by CHAR_e we have $s \in_e r$. In the second case, $r = r_1^*$ for some r_1 and $s = \varepsilon$. By STAR-1_e we have $s \in_e r$.

   If neither of those rules applies, the derivation tree has height $n > 1$ and must have one of the other four rules at the bottom. Consider the four possibilities for the last rule.

   - CONCAT_a: Then $s = s_1 s_2$, $r = r_1 \cdot r_2$, $s_1 \in_a r_1$ and $s_2 \in_a r_2$. By the induction hypothesis, $s_1 \in_e r_1$ and $s_2 \in_e r_2$. Applying the rule CONCAT_1 gives a derivation for $s \in_e r$.
   - UNION-1_a or UNION-1_a: The reasoning for these cases is similar to the reasoning for the previous case.
   - STAR-2_a: Then $s = s_1 s_2$, $r = r_1^*$, and $s_1 \in_a r_1$ and $s_2 \in_a r_1^*$. These last two derivations are shorter than the original derivation so we can apply the induction hypothesis to conclude $s_1 \in_e r_1$ and $s_2 \in_e r_1^*$. Suppose $s_1 \neq \varepsilon$. Then applying the rule STAR-2_e gives a derivation for $s \in_e r$. Now suppose $s_1 = \varepsilon$. Then $s = s_1 s_2 = s_2$ and the derivation $s_2 \in_e r_1^*$ is a derivation for $s \in_e r$ and we are done.

   Now suppose there is a derivation $s \in_e r$ which uses the semantics from part (e). This is shown by induction on the height of the derivation $s \in_e r$. For all cases but STAR-2_e, the reasoning is the same as in the other direction, swapping references to the part (a) semantics and the part (e) semantics.

   For the case STAR-2_e, suppose $s \in_e r$ then $s = s_1 s_2$, $r = r_1^*$, $s_1 \in_e r_1$ and $s_2 \in_e r_1^*$. By the induction hypothesis, $s_1 \in_a r_1$ and $s_2 \in_a r_1^*$. Applying the rules STAR-2_a gives a derivation for $s \in_a r$.

   Having shown both directions of the proof, we can conclude the theorem holds.

   □

(f) See `interp.ml`.

2. Locks and simulating non-pre-emption

(a) Small-step operational semantics for IMP with locks.

$\boxed{H\ ;\ s\ ;\ q;\ L \rightarrow H;\ s\ ;\ q\ ;\ L\ ;\ b}$

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

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

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

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

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

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

IF2
$$\frac{H\ ;\ e\ \Downarrow\ c \qquad c \leq 0}{H\ ;\ \mathsf{if}\ e\ s_1\ s_2\ ;\ q;\ L \rightarrow H;\ s_2\ ;\ q\ ;\ L\ ;\ \mathsf{false}}$$

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

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

PRE-EMPT
$$\frac{}{H\ ;\ s\ ;\ s'{::}q;\ L \rightarrow H;\ s'\ ;\ q{::}s\ ;\ L\ ;\ \mathsf{true}}$$

ACQUIRE
$$\frac{c \notin L}{H\ ;\ \mathsf{acquire}(c)\ ;\ s{::}q;\ L \rightarrow H;\ s\ ;\ q{::}\mathsf{skip}\ ;\ L \cup \{c\}\ ;\ \mathsf{true}}$$

RELEASE
$$\frac{c \in L}{H\ ;\ \mathsf{release}(c)\ ;\ s{::}q;\ L \rightarrow H;\ s\ ;\ q{::}\mathsf{skip}\ ;\ L - \{c\}\ ;\ \mathsf{true}}$$

(b) We will use two function, ts to translate individual statements and tt to translate whole programs or threads.

$$tt(s) \quad = \quad \mathsf{acquire}(0); s; \mathsf{release}(0)$$

$$\begin{array}{lcl} ts(\mathsf{yield}) & = & \mathsf{release}(0); \mathsf{acquire}(0) \\ ts((spawn)(s)) & = & \mathsf{spawn}(tt(s)) \\ ts(\mathsf{if}\ e\ s_1\ s_2) & = & \mathsf{if}\ e\ ts(s_1)\ ts(s_2) \\ ts(\mathsf{while}\ e\ s) & = & \mathsf{while}\ e\ ts(s) \\ ts(x := e) & = & \mathsf{x} := \mathsf{e} \\ ts(\mathsf{skip}) & = & \mathsf{skip} \end{array}$$

(c) In the unmodified source language, if a thread yielded, it would *always* be put on the queue and the thread on the front of the queue would be the next to be run. That cannot be simulated in

our language with locks because pre-emption is always optional. That is, when we release a lock, we may be pre-empted, but we may also immediately acquire the lock again.

This question did not ask why the source language needed to have non-deterministic scheduling added, but here is why: When a thread yields in the unmodified source language the next thread on the queue is guaranteed to be the next thread executed. In our language with locks, if you release a lock and are pre-empted, it may not be the case that the next thread on the queue is always the next thread run. The next thread on the queue may be pre-empted before it can ever acquire the lock.