1 3F-1 Bookkeeping

- **0 pts** Correct

Exercise 3F-2. Regular Expression, Large-Step [10 points]. The large step operational semantics for concatenation, or, and Kleene star of regular expressions are as follows.

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } s' \quad \vdash e_2 \text{ matches } s' \text{ leaving } s''}{\vdash e_1 e_2 \text{ matches } s \text{ leaving } s''}$$

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } s'}{\vdash e_1 | e_2 \text{ matches } s \text{ leaving } s'}$$

$$\frac{\vdash e_2 \text{ matches } s \text{ leaving } s'}{\vdash e_1 | e_2 \text{ matches } s \text{ leaving } s'}$$

$$\frac{}{\vdash e^* \text{ matches } s \text{ leaving } s}$$

$$\frac{\vdash e \text{ matches } s \text{ leaving } s' \quad \vdash e^* \text{ matches } s' \text{ leaving } s''}{\vdash e^* \text{ matches } s \text{ leaving } s''}$$

Exercise 3F-3. Regular Expression and Sets [5 points]. $e_1 e_2$ cannot be expressed with the given framework. You could try to do:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S \quad \vdash e_2 \text{ matches } S \text{ leaving } S'}{\vdash e_1 e_2 \text{ matches } s \text{ leaving } S'}$$

However, S is a set, so you would need a judgement that matches all strings in a set and leaves a set, which isn't provided in the framework. Similarly for e^*, you could try to do:

$$\frac{\vdash e \text{ matches } s \text{ leaving } S \quad \vdash e^* \text{ matches } S \text{ leaving } S'}{\vdash e^* \text{ matches } s \text{ leaving } S'}$$

but you run into the same problem. Thus, both of these rules are incomplete.

Exercise 3F-4. Equivalence [7 points]. Assume that $e_1 \sim e_2$ is decidable. Then we can define an algorithm isEquivalent(e_1, e_2) that takes two regular expressions and returns true if they are equivalent and false if anything else happens. This algorithm must return in finite time by the definition of an algorithm. Next, define the following program that executes someCode then returns a regular expression:

```
function getRegExpr(someCode):
    someCode()
    return e
```

Now we call isEquivalent and pass in code that solves the halting problem.

```
isEquivalent(getRegExpr(haltingProblemInput), getRegExpr(haltingProblemInput))
```

Since isEquivalent must return in finite time, this means that the halting problem was solved. This is impossible, so $e_1 \sim e_2$ is undecideable.

2 3F-2 Regular Expressions, Large Step

- **0 pts** Correct

Exercise 3F-2. Regular Expression, Large-Step [10 points]. The large step operational semantics for concatenation, or, and Kleene star of regular expressions are as follows.

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } s' \quad \vdash e_2 \text{ matches } s' \text{ leaving } s''}{\vdash e_1 e_2 \text{ matches } s \text{ leaving } s''}$$

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } s'}{\vdash e_1 | e_2 \text{ matches } s \text{ leaving } s'}$$

$$\frac{\vdash e_2 \text{ matches } s \text{ leaving } s'}{\vdash e_1 | e_2 \text{ matches } s \text{ leaving } s'}$$

$$\frac{}{\vdash e^* \text{ matches } s \text{ leaving } s}$$

$$\frac{\vdash e \text{ matches } s \text{ leaving } s' \quad \vdash e^* \text{ matches } s' \text{ leaving } s''}{\vdash e^* \text{ matches } s \text{ leaving } s''}$$

Exercise 3F-3. Regular Expression and Sets [5 points]. $e_1 e_2$ cannot be expressed with the given framework. You could try to do:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S \quad \vdash e_2 \text{ matches } S \text{ leaving } S'}{\vdash e_1 e_2 \text{ matches } s \text{ leaving } S'}$$

However, S is a set, so you would need a judgement that matches all strings in a set and leaves a set, which isn't provided in the framework. Similarly for e^*, you could try to do:

$$\frac{\vdash e \text{ matches } s \text{ leaving } S \quad \vdash e^* \text{ matches } S \text{ leaving } S'}{\vdash e^* \text{ matches } s \text{ leaving } S'}$$

but you run into the same problem. Thus, both of these rules are incomplete.

Exercise 3F-4. Equivalence [7 points]. Assume that $e_1 \sim e_2$ is decidable. Then we can define an algorithm isEquivalent(e_1, e_2) that takes two regular expressions and returns true if they are equivalent and false if anything else happens. This algorithm must return in finite time by the definition of an algorithm. Next, define the following program that executes someCode then returns a regular expression:

```
function getRegExpr(someCode):
    someCode()
    return e
```

Now we call isEquivalent and pass in code that solves the halting problem.

```
isEquivalent(getRegExpr(haltingProblemInput), getRegExpr(haltingProblemInput))
```

Since isEquivalent must return in finite time, this means that the halting problem was solved. This is impossible, so $e_1 \sim e_2$ is undecideable.

3 3F-3 Regular Expressions and Sets

- 0 pts Correct

Exercise 3F-2. Regular Expression, Large-Step [10 points]. The large step operational semantics for concatenation, or, and Kleene star of regular expressions are as follows.

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } s' \quad \vdash e_2 \text{ matches } s' \text{ leaving } s''}{\vdash e_1 e_2 \text{ matches } s \text{ leaving } s''}$$

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } s'}{\vdash e_1 | e_2 \text{ matches } s \text{ leaving } s'}$$

$$\frac{\vdash e_2 \text{ matches } s \text{ leaving } s'}{\vdash e_1 | e_2 \text{ matches } s \text{ leaving } s'}$$

$$\frac{}{\vdash e^* \text{ matches } s \text{ leaving } s}$$

$$\frac{\vdash e \text{ matches } s \text{ leaving } s' \quad \vdash e^* \text{ matches } s' \text{ leaving } s''}{\vdash e^* \text{ matches } s \text{ leaving } s''}$$

Exercise 3F-3. Regular Expression and Sets [5 points]. $e_1 e_2$ cannot be expressed with the given framework. You could try to do:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S \quad \vdash e_2 \text{ matches } S \text{ leaving } S'}{\vdash e_1 e_2 \text{ matches } s \text{ leaving } S'}$$

However, S is a set, so you would need a judgement that matches all strings in a set and leaves a set, which isn't provided in the framework. Similarly for e^*, you could try to do:

$$\frac{\vdash e \text{ matches } s \text{ leaving } S \quad \vdash e^* \text{ matches } S \text{ leaving } S'}{\vdash e^* \text{ matches } s \text{ leaving } S'}$$

but you run into the same problem. Thus, both of these rules are incomplete.

Exercise 3F-4. Equivalence [7 points]. Assume that $e_1 \sim e_2$ is decidable. Then we can define an algorithm isEquivalent(e_1, e_2) that takes two regular expressions and returns true if they are equivalent and false if anything else happens. This algorithm must return in finite time by the definition of an algorithm. Next, define the following program that executes someCode then returns a regular expression:

```
function getRegExpr(someCode):
    someCode()
    return e
```

Now we call isEquivalent and pass in code that solves the halting problem.

```
isEquivalent(getRegExpr(haltingProblemInput), getRegExpr(haltingProblemInput))
```

Since isEquivalent must return in finite time, this means that the halting problem was solved. This is impossible, so $e_1 \sim e_2$ is undecideable.

4 3F-4 Equivalence

- **0 pts** Correct

Exercise 3F-5. SAT Solving [6 points]. The last two included tests take a much longer time than the other tests because they have three arithmetic variables, whereas the other tests have at most two. The arithmetic solver tries assignments of every integer from -127 to 128 inclusive, so there are 256 possible assignments per arithmetic variable. This makes the complexity of the solver $O(256^n)$, where n is the number of arithmetic variables. Thus, as the number of arithmetic variables grows, the running time of the arithmetic solver grows very quickly, and it starts to become noticeable when there are three. To improve this, we could change the bounded search module to plot the constraints in Cartesian coordinates and use hill-climbing to check the local maximums of the area that satisfies all constraints. This would make the assignment part of the algorithm $O(n)$, which is much faster.

The limited search from -127 to 128 is a major defect. This arithmetic solver would say that $x < -127$ is unsatisfiable, but it obviously is satisfiable with $x = -128$.

5 3F-5 SAT Solving

- **0 pts** Correct