1 3F-1 Bookkeeping

- **0 pts** Correct

Exercise 3F-2. Regular Expression, Large-Step

Large-step Operational Semantics Rule of Inference for Concatenation:

$$\frac{\vdash e1 \; matches \; s \; leaving \; s1 \qquad \vdash e2 \; matches \; s1 \; leaving \; s2}{\vdash e1 \; e2 \; matches \; s \; leaving \; s2}$$

Large-step Operational Semantics Rule of Inference for Or:

$$\frac{\vdash e1 \; matches \; s \; leaving \; s1}{\vdash e1 \mid e2 \; matches \; s \; leaving \; s1}$$

$$\frac{\vdash e2 \; matches \; s \; leaving \; s1}{\vdash e1 \mid e2 \; matches \; s \; leaving \; s1}$$

Large-step Operational Semantics Rule of Inference for Kleene Star:

$$\frac{}{\vdash e* \; matches \; s \; leaving \; s}$$

$$\frac{\vdash e \; matches \; s \; leaving \; s1 \qquad \vdash e* \; matches \; s1 \; leaving \; s2}{\vdash e* \; matches \; s \; leaving \; s2}$$

2 3F-2 Regular Expressions, Large Step

- **0 pts** Correct

Exercise 3F-3. Regular Expression and Sets

It cannot be done correctly with this given framework. For e1 e2, e1 has to be matched before e2 to be matched, so it is impossible to be done without nesting derivations in set constructors as e1 is required to be evaluated first. Therefore, the rule for e1 e2 has to use a derivation inside of a set constructor, but it is incomplete. This wrong rule is present below.

$$\frac{\vdash e1 \ matches \ s \ leaving \ S \qquad \vdash e2 \ matches \ S \ leaving \ S1 = \{s1 \mid \exists \ s \in S \vdash e2 \ matches \ s \ leaving \ s1\}}{\vdash e1 \ e2 \ matches \ s \ leaving \ S1}$$

To avoid using a derivation inside of a set constructor, it is necessary to come up with a rule that ensures evaluation of expressions in order like below.

$$\frac{\vdash e1 \ matches \ s \ leaving \ S1 \qquad \vdash e2 \ matches \ s \ leaving \ S2}{\vdash e1 \ e2 \ matches \ s \ leaving \ S1 \cap S2}$$

As shown on the above rule, it does not use a derivation inside a set constructor. However, it will return incorrect results because it can only check whether e1 and e2 both are matched in s, but not in consecutive order.

3 3F-3 Regular Expressions and Sets

- **0 pts** Correct

Exercise 3F-4. Equivalence

e1 ~ e2 is undecidable, and I will prove it by reducing it to the halting problem, which is an undecidable problem. For the purpose of contradiction, assume that if I could solve “e1 ~ e2”, then I could also solve the halting problem. Assume that there is a solver for “e1 ~ e2”, so I have an algorithm called ISEQUAL(a, b) that takes in two descriptions of regular expressions and returns true if they are the same on all string inputs. Now, I will try to solve the halting problem. Suppose that there is an arbitrary source code instance called “hproblem_input” which returns true if and only if “hproblem_input” terminates in finite time. Then, I will construct a finite time program called PQR as below:

```
def PQR():
    hproblem_input();
    return e
```

Which returns a regular expression e if and only if “hproblem_input” terminates in finite time. Now I will invoke ISEQUAL(PQR, PQR). Because ISEQUAL is supposed to always make a correct decision in finite time if two descriptions of regular expressions are the same and because PQR returns regular expression e if “hproblem_input” terminates in finite time, it implies that ISEQUAL can actually solve for whether “hproblem_input” halts or not. However, the halting problem is known to be impossible to solve. Therefore, because of this contradiction, it is impossible to solve “e1 ~ e2” in finite time which indicates that it is undecidable.

4 3F-4 Equivalence

- **0 pts** Correct

Exercise 3F-5. SAT Solving

The last two included tests take a comparatively long time because DPLL(T) is a backtracking-based search algorithm. The two test cases contain comparatively more inequality evaluations compared to other test cases, so the DPLL(T) solver needs to explore a greater search space to figure out whether an input formula is satisfiable or not. This greater search space in the last two test cases might impact the running time and efficiency of DPLL(T) solver. In the code, I would rewrite the arith module first to improve performance because it currently tries out all the possibility of integer valuations to all variables in the constraints on the given range bounded by lower_bound and upper_bound. Instead of exhaustive linear search within the bound, I would change it to binary search so that its running time complexity is improved. Or alternatively, because the arith module currently runs in exponential time, I will change the arith module by adopting some method to run in polynomial time instead.

5 3F-5 SAT Solving

- **0 pts** Correct