1 3F-1 Bookkeeping

- **0 pts** Correct

Exercise 3F-2

Concatenation:

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

Or:

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

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

Kleene star:

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

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

2 3F-2 Regular Expressions, Large Step

- **0 pts** Correct

Exercise 3F-3

I don't think we can form operational semantic rules for $e*$ or e_1e_2 correctly without placing derivation inside set constructors. Both $e*$ and e_1e_2 have reasonings involving using the output of the previous hypothesis as the input of the next hypothesis, which would result in a unfixed set of hypotheses in order to represent all possible answers as requested in the question. We would indeed attempt to represent some cases of how $e*$ and e_1e_2 operate, but those inference rules will be incomplete.

$$\frac{\vdash e_1 \ matches \ s \ leaving \ s' \quad \vdash e_2 \ matches \ s' \ leaving \ s''}{\vdash e_1e_2 \ matches \ s \ leaving \ s''}$$

Taken the above concatenation rule as an example, without using syntaxes like

$$\frac{\vdash e_1 \ matches \ s \ leaving \ S' \quad \vdash s_1 \in S' \ \vdash e_2 \ matches \ s_1 \ leaving \ S_1 ... \ \vdash s_n \in S' \ \vdash e_2 \ matches \ s_n \ leaving \ S_n}{\vdash e_1e_2 \ matches \ s \ leaving \ S_1 \cup S_2 ... \cup S_n}$$

, we are only representing one case of $e_1 \ matches \ s$ with its output s' passing into the next hypothesis. This operational semantics is incomplete because it fails to include/prove all true scenarios of the concatenation of regular expression.

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

The Kleene star rule above is also incomplete as it doesn't capture the case where e^* is an empty string and it only represents one case of $e_1 \ matches \ s$ with its output s' passing into the next hypothesis.

Exercise 3F-4

The equivalence between regular expressions, $e_1 \sim e_2$, is decidable because we can reduce a regular expression to a DFA(Deterministic Finite Automaton), and prove that DFA equivalence is decidable.

More specifically, we can generate NFAs (Non-deterministic Finite Automaton) for regular expressions e_1, e_2 as they share the same expression power. Then we further transfer those two NFAs to DFAs. After reducing DFAs into their minimal form, testing for equivalence will be decidable since the minimal form of a DFA is unique up to isomorphism. Because the states of the minimal DFAs are bounded by Cartesian product machine construction, we can try all strings of length up to the bounded state number then prove the languages that the two DFAs represented are the subset of each other. Hence, they are equivalent.

3 3F-3 Regular Expressions and Sets

- **0 pts** Correct

Exercise 3F-3

I don't think we can form operational semantic rules for $e*$ or e_1e_2 correctly without placing derivation inside set constructors. Both $e*$ and e_1e_2 have reasonings involving using the output of the previous hypothesis as the input of the next hypothesis, which would result in a unfixed set of hypotheses in order to represent all possible answers as requested in the question. We would indeed attempt to represent some cases of how $e*$ and e_1e_2 operate, but those inference rules will be incomplete.

$$\frac{\vdash e_1 \ matches \ s \ leaving \ s' \quad \vdash e_2 \ matches \ s' \ leaving \ s''}{\vdash e_1e_2 \ matches \ s \ leaving \ s''}$$

Taken the above concatenation rule as an example, without using syntaxes like

$$\frac{\vdash e_1 \ matches \ s \ leaving \ S' \quad \vdash s_1 \in S' \ \vdash e_2 \ matches \ s_1 \ leaving \ S_1 ... \ \vdash s_n \in S' \ \vdash e_2 \ matches \ s_n \ leaving \ S_n}{\vdash e_1e_2 \ matches \ s \ leaving \ S_1 \cup S_2 ... \cup S_n}$$

, we are only representing one case of $e_1 \ matches \ s$ with its output s' passing into the next hypothesis. This operational semantics is incomplete because it fails to include/prove all true scenarios of the concatenation of regular expression.

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

The Kleene star rule above is also incomplete as it doesn't capture the case where e^* is an empty string and it only represents one case of $e_1 \ matches \ s$ with its output s' passing into the next hypothesis.

Exercise 3F-4

The equivalence between regular expressions, $e_1 \sim e_2$, is decidable because we can reduce a regular expression to a DFA(Deterministic Finite Automaton), and prove that DFA equivalence is decidable.

More specifically, we can generate NFAs (Non-deterministic Finite Automaton) for regular expressions e_1, e_2 as they share the same expression power. Then we further transfer those two NFAs to DFAs. After reducing DFAs into their minimal form, testing for equivalence will be decidable since the minimal form of a DFA is unique up to isomorphism. Because the states of the minimal DFAs are bounded by Cartesian product machine construction, we can try all strings of length up to the bounded state number then prove the languages that the two DFAs represented are the subset of each other. Hence, they are equivalent.

4 3F-4 Equivalence

- **0 pts** Correct

Exercise 3F-6

Test 35, 36 are slower than the rest because they involve more variables than others. In the Arithmetic Solver, the code exhaustively loops through all possible values of each variable to find a match that solves the equalities. This would result in an exponentially increasing complexity when the number of variables in input increases. To improve this inefficiency, the algorithm should generate a bounded range of each variable based on the equalities given. Here the code should ensure the resulting bounds are as tight as possible in respect to the intersection of different equalities. Then it can use binary search, instead of natively looping through, to backtrace the target value that satisfies all the equalities.

5 3F-5 SAT Solving

- **0 pts** Correct