1 3F-1 Bookkeeping

- 0 pts Correct

Exercise 3F-1. Regular Expression, Large-Step [10 points].
Concatenation – matches e_1 followed by e_2:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } s_0 \quad \vdash e_2 \text{ matches } s_0 \text{ leaving } s'}{\vdash e_1\ e_2 \text{ matches } s \text{ leaving } s'}$$

or – matches e_1 or e_2:

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

Kleene star – matches 0 or more occurences of e:

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

Exercise 3F-2. Regular Expression and Sets [5 points].
I claim that we *cannot* construct operational semantics rules of inference for $e*$ and e_1e_2 in the given framework because such rules would require either derivations inside of set constructors or a set of hypothesis that vary depending on the s in question (i.e. are not fixed and finite). See the following attempted but "wrong" rules of inference:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S_0 \quad \forall s_i \in S_0. \vdash e_2 \text{ matches } s_i \text{ leaving } S_i \quad S = \bigcup_{i=1} S_i}{\vdash e_1\ e_2 \text{ matches } s \text{ leaving } S}$$

This rule first matches e_1 to get an exhaustive set of suffices S_0, then applies e_2 to each suffix s_i in S_0 to get an exhaustive set of suffices S_i. Intuitively, e_1e_2 should map to the union of possible suffices from maching e_1, then e_2, and this intuitive notion of regular expression matching is captured by the above rule. BUT, as the number of elements in S_0 is *not fixed* (can range from 0 to the cardinality of s depending on s and e_1), this rule is not valid.

$$\frac{\vdash e^* \text{ matches } s \text{ leaving } \{s \mid s_n = e^* :: s'\}}{\vdash e^* \text{ matches } s \text{ leaving } S}$$

Similar to the rules in 3F-2, this rule tries to capture the idea that the set of possible suffixes ranges from the entire string (match to 0 occurrences) to as many repeated matches of e are possible. BUT, this rule is not valid as it is a type error to concatenate a regular expression to a string, so s_n is not a valid suffix and this rule fails to express anything meaningful.

Exercise 3F-3. Equivalence [7 points].
I claim that $e_1 \sim e_2$ is *decidable*. Intuitively, regular expressions seem to be capturing a syntactic and not a symantic property which should make them decidable; formally, we define an algorithm for deciding whether $e_1 \sim e_2$ that proceeds as follows:

1. Convert e_1 and e_2 into a composition of *primary* regular expression forms (simplifying . and ["x" - "y"] to *or* statements of singletons; e? to empty | e; and e+ to e e*).

2 3F-2 Regular Expressions, Large Step

- 0 pts Correct

Exercise 3F-1. Regular Expression, Large-Step [10 points].
Concatenation – matches e_1 followed by e_2:

$$\frac{\vdash e_1 \textsf{ matches } s \textsf{ leaving } s_0 \quad \vdash e_2 \textsf{ matches } s_0 \textsf{ leaving } s'}{\vdash e_1\ e_2 \textsf{ matches } s \textsf{ leaving } s'}$$

or – matches e_1 or e_2:

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

Kleene star – matches 0 or more occurences of e:

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

Exercise 3F-2. Regular Expression and Sets [5 points].
I claim that we *cannot* construct operational semantics rules of inference for $e*$ and $e_1 e_2$ in the given framework because such rules would require either derivations inside of set constructors or a set of hypothesis that vary depending on the s in question (i.e. are not fixed and finite). See the following attempted but "wrong" rules of inference:

$$\frac{\vdash e_1 \textsf{ matches } s \textsf{ leaving } S_0 \quad \forall s_i \in S_0. \vdash e_2 \textsf{ matches } s_i \textsf{ leaving } S_i \quad S = \bigcup_{i=1} S_i}{\vdash e_1\ e_2 \textsf{ matches } s \textsf{ leaving } S}$$

This rule first matches e_1 to get an exhaustive set of suffices S_0, then applies e_2 to each suffix s_i in S_0 to get an exhaustive set of suffices S_i. Intuitively, $e_1 e_2$ should map to the union of possible suffices from maching e_1, then e_2, and this intuitive notion of regular expression matching is captured by the above rule. BUT, as the number of elements in S_0 is *not fixed* (can range from 0 to the cardinality of s depending on s and e_1), this rule is not valid.

$$\frac{\vdash e^* \textsf{ matches } s \textsf{ leaving } \{s \mid s_n = e^* :: s'\}}{\vdash e^* \textsf{ matches } s \textsf{ leaving } S}$$

Similar to the rules in 3F-2, this rule tries to capture the idea that the set of possible suffixes ranges from the entire string (match to 0 occurrences) to as many repeated matches of e are possible. BUT, this rule is not valid as it is a type error to concatenate a regular expression to a string, so s_n is not a valid suffix and this rule fails to express anything meaningful.

Exercise 3F-3. Equivalence [7 points].
I claim that $e_1 \sim e_2$ is *decidable*. Intuitively, regular expressions seem to be capturing a syntactic and not a symantic property which should make them decidable; formally, we define an algorithm for deciding whether $e_1 \sim e_2$ that proceeds as follows:

1. Convert e_1 and e_2 into a composition of *primary* regular expression forms (simplifying . and ["x" - "y"] to *or* statements of singletons; e? to empty | e; and e+ to e e*).

3 3F-3 Regular Expressions and Sets

- 0 pts Correct

Exercise 3F-1. Regular Expression, Large-Step [10 points].
Concatenation – matches e_1 followed by e_2:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } s_0 \quad \vdash e_2 \text{ matches } s_0 \text{ leaving } s'}{\vdash e_1\ e_2 \text{ matches } s \text{ leaving } s'}$$

or – matches e_1 or e_2:

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

Kleene star – matches 0 or more occurences of e:

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

Exercise 3F-2. Regular Expression and Sets [5 points].
I claim that we *cannot* construct operational semantics rules of inference for $e*$ and $e_1 e_2$ in the given framework because such rules would require either derivations inside of set constructors or a set of hypothesis that vary depending on the s in question (i.e. are not fixed and finite). See the following attempted but "wrong" rules of inference:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S_0 \quad \forall s_i \in S_0 . \vdash e_2 \text{ matches } s_i \text{ leaving } S_i \quad S = \bigcup_{i=1} S_i}{\vdash e_1\ e_2 \text{ matches } s \text{ leaving } S}$$

This rule first matches e_1 to get an exhaustive set of suffices S_0, then applies e_2 to each suffix s_i in S_0 to get an exhaustive set of suffices S_i. Intuitively, $e_1 e_2$ should map to the union of possible suffices from maching e_1, then e_2, and this intuitive notion of regular expression matching is captured by the above rule. BUT, as the number of elements in S_0 is *not fixed* (can range from 0 to the cardinality of s depending on s and e_1), this rule is not valid.

$$\frac{\vdash e^* \text{ matches } s \text{ leaving } \{s \mid s_n = e^* :: s'\}}{\vdash e^* \text{ matches } s \text{ leaving } S}$$

Similar to the rules in 3F-2, this rule tries to capture the idea that the set of possible suffixes ranges from the entire string (match to 0 occurrences) to as many repeated matches of e are possible. BUT, this rule is not valid as it is a type error to concatenate a regular expression to a string, so s_n is not a valid suffix and this rule fails to express anything meaningful.

Exercise 3F-3. Equivalence [7 points].
I claim that $e_1 \sim e_2$ is *decidable*. Intuitively, regular expressions seem to be capturing a syntactic and not a symantic property which should make them decidable; formally, we define an algorithm for deciding whether $e_1 \sim e_2$ that proceeds as follows:

1. Convert e_1 and e_2 into a composition of *primary* regular expression forms (simplifying . and ["x" - "y"] to *or* statements of singletons; e? to empty | e; and e+ to e e*).

2. Eliminate redudant expressions and simplify expressions as much as possible.

3. Return "equal" if the simplified forms of e_1 and e_2 are equivalent (where order of or regexes does *not* matter) and "not equal" otherwise (as the regexes aren't equal after simplification, there must exist some string s for which $\vdash e_1$ matches s leaving $S_1 \wedge \vdash e_2$ matches s leaving S_2 but $S_1 \neq S_2$).

Exercise 3C. SAT Solving. See submission on autograder.io.

Exercise 3F-4. SAT Solving [6 points].
The last two included tests took much longer because they include more complicated inequalities across variables that introduce dependencies, add mixed constraints that could preclude the SAT solver from applying certain heuristics in some cases, and complicate the amount of effort required by the theory.

To improve performance, I would first optimize the theory solver (arth.ml) to improve the simple but horridly inefficient bounded_search integer constraint solver. Ganziner et. al. mention an efficient integration of specialized theory solvers within a general purpose engine, and the specialized theory solver is currently inefficiently integrated. Instead of trying all possible variables within the bounds, I would at minimum take the constraints into account to avoiding needlessly searching unnecessary assignments when there are more than one variable and clear constraints, and strongly consider replacing bounded search with another integer constrain solving method altogether.

Comment: considering all possible integer valutions to all variables in the constraints, but only working with integer variables between -127 to 128 is not the behavior we're wanting from our solver.

4 3F-4 Equivalence

- 0 pts Correct

2. Eliminate redudant expressions and simplify expressions as much as possible.

3. Return “equal” if the simplified forms of e_1 and e_2 are equivalent (where order of or regexes does *not* matter) and “not equal” otherwise (as the regexes aren’t equal after simplification, there must exist some string s for which $\vdash e_1 \textsf{ matches } s \textsf{ leaving } S_1 \land \vdash e_2 \textsf{ matches } s \textsf{ leaving } S_2$ but $S_1 \neq S_2$).

Exercise 3C. SAT Solving. See submission on autograder.io.

Exercise 3F-4. SAT Solving [6 points].
The last two included tests took much longer because they include more complicated inequalities across variables that introduce dependencies, add mixed constraints that could preclude the SAT solver from applying certain heuristics in some cases, and complicate the amount of effort required by the theory.

To improve performance, I would first optimize the theory solver (arth.ml) to improve the simple but horridly inefficient bounded_search integer constraint solver. Ganziner et. al. mention an efficient integration of specialized theory solvers within a general purpose engine, and the specialized theory solver is currently inefficiently integrated. Instead of trying all possible variables within the bounds, I would at minimum take the constraints into account to avoiding needlessly searching unnecessary assignments when there are more than one variable and clear constraints, and strongly consider replacing bounded search with another integer constrain solving method altogether.

Comment: considering all possible integer valutions to all variables in the constraints, but only working with integer variables between -127 to 128 is not the behavior we’re wanting from our solver.

5 3F-5 SAT Solving

- 0 pts Correct