1 3F-1 Bookkeeping

- **0 pts** Correct

Exercise 3F-1. Regular Expression, Large-Step

$$\frac{\vdash e_1 \text{ matches } s_1 \text{ leaving } s_2 \qquad \vdash e_2 \text{ matches } s_2 \text{ leaving } s_3}{\vdash e_1 e_2 \text{ matches } s_1 \text{ leaving } s_3}$$

$$\frac{\vdash e_1 \text{ matches } s_1 \text{ leaving } s_2}{\vdash e_1 | e_2 \text{ matches } s_1 \text{ leaving } s_2}$$

$$\frac{\vdash e_2 \text{ matches } s_1 \text{ leaving } s_2}{\vdash e_1 | e_2 \text{ matches } s_1 \text{ leaving } s_2}$$

$$\frac{}{\vdash e* \text{ matches } s_1 \text{ leaving } s_1}$$

$$\frac{\vdash e \text{ matches } s_1 \text{ leaving } s_2 \qquad \vdash e* \text{ matches } s_2 \text{ leaving } s_3}{\vdash e* \text{ matches } s_1 \text{ leaving } s_3}$$

Exercise 3F-2. Regular Expression and Sets

We **can't** form operaitonal semantics rules for $e*$ or $e_1 e_2$ given the current framework. The key difference of such primary forms from those for which **can** construct operational semantics rules (e.g. " x ", empty, $e_1 | e_2$) is that

1. They contain other primary forms as their sub-parts.
2. The input of one sub-part is taken from the output set of another sub-part.

Take $e_1 e_2$ as an example, the input of e_2 is taken from the output set of e_1, so if we will have to construct a rule, it will look like

$$\frac{\vdash e_1 \text{ matches } s_1 \text{ leaving } S_1 \qquad \vdash e_2 \text{ matches } s_2 \text{ leaving } S_2 \qquad \vdash s_2 \in S_1}{\vdash e_1 e_2 \text{ matches } s_1 \text{ leaving } S_2}$$

Unfortunately, this is incomplete, because s_2 is not able to capture all elements in S_1. Similarly for $e*$, if we have to construct a rule, it would look like

$$\frac{\vdash e \text{ matches } s_1 \text{ leaving } S_1 \qquad \vdash e* \text{ matches } s_2 \text{ leaving } S_2 \qquad \vdash s_2 \in S_1}{\vdash e* \text{ matches } s_1 \text{ leaving } S_2 \cup \{s_1\}}$$

but this is incomplete too.

2 3F-2 Regular Expressions, Large Step

- **0 pts** Correct

Exercise 3F-1. Regular Expression, Large-Step

$$\frac{\vdash e_1 \text{ matches } s_1 \text{ leaving } s_2 \qquad \vdash e_2 \text{ matches } s_2 \text{ leaving } s_3}{\vdash e_1 e_2 \text{ matches } s_1 \text{ leaving } s_3}$$

$$\frac{\vdash e_1 \text{ matches } s_1 \text{ leaving } s_2}{\vdash e_1 | e_2 \text{ matches } s_1 \text{ leaving } s_2}$$

$$\frac{\vdash e_2 \text{ matches } s_1 \text{ leaving } s_2}{\vdash e_1 | e_2 \text{ matches } s_1 \text{ leaving } s_2}$$

$$\frac{}{\vdash e* \text{ matches } s_1 \text{ leaving } s_1}$$

$$\frac{\vdash e \text{ matches } s_1 \text{ leaving } s_2 \qquad \vdash e* \text{ matches } s_2 \text{ leaving } s_3}{\vdash e* \text{ matches } s_1 \text{ leaving } s_3}$$

Exercise 3F-2. Regular Expression and Sets

We **can't** form operaitonal semantics rules for $e*$ or $e_1 e_2$ given the current framework. The key difference of such primary forms from those for which **can** construct operational semantics rules (e.g. " x ", empty, $e_1 | e_2$) is that

1. They contain other primary forms as their sub-parts.
2. The input of one sub-part is taken from the output set of another sub-part.

Take $e_1 e_2$ as an example, the input of e_2 is taken from the output set of e_1, so if we will have to construct a rule, it will look like

$$\frac{\vdash e_1 \text{ matches } s_1 \text{ leaving } S_1 \qquad \vdash e_2 \text{ matches } s_2 \text{ leaving } S_2 \qquad \vdash s_2 \in S_1}{\vdash e_1 e_2 \text{ matches } s_1 \text{ leaving } S_2}$$

Unfortunately, this is incomplete, because s_2 is not able to capture all elements in S_1. Similarly for $e*$, if we have to construct a rule, it would look like

$$\frac{\vdash e \text{ matches } s_1 \text{ leaving } S_1 \qquad \vdash e* \text{ matches } s_2 \text{ leaving } S_2 \qquad \vdash s_2 \in S_1}{\vdash e* \text{ matches } s_1 \text{ leaving } S_2 \cup \{s_1\}}$$

but this is incomplete too.

3 3F-3 Regular Expressions and Sets

- **0 pts** Correct

Exercise 3F-3. Equivalence

Regular expression equivalence $e_1 \sim e_2$ is decidable, because a regular expression equivalence can be reduced to DFA (Deterministic Finite Automaton) equivalence and DFA equivalence is decidable.

We can translate a regular expression into an equivalent DFA by **first translating it into an equivalent NFA** (Non-deterministic Finite Automaton) with Thompson's construction, and **then to DFA** with subset construction algorithm. Now to show regular expression equivalence is decidable, we only need to show that FDA equivalence is decidable.

Let A and B be two arbitrary DFA. A and B are equivalent if and only if A and B accept the same strings up to length $m * n$, where m and n are the number of states in A and B. This is because if A and B are not equivalent, they must output different results for some string of length at most $m * n$. Since the number of strings of length up to $m * n$ is finite, we can solve this problem in finite amount of time. i.e. DFA equivalence is decidable.

Ref: https://cs.stackexchange.com/questions/92496/proving-that-dfa-equivalence-is-decidable

Exercise 3F-4. SAT Solving

The test 35 and 36 are slow because they have 3 variables while the rest of the test have at most 2 variables. This matters a lot because in the given code, the **arithmetic solver** works in a way that generates all possible combination of variables ranging from -127 to 128 and naively checks all possible combinations against conditions. This means for a input with n variables, the arithmetic solver takes $O(256^n)$ times to execute.

The performance could be made much better if we rewrite the arithmetic solver into a **linear programming solver**. Instead of sweeping through the whole variable space, we could focus on just the intersections of functions defined by linear inequalities. We could further improve by focusing on intersections that involves all the equalities. Take test case 26 $(x > y)\&\&(y > z)\&\&(z = 10)\&\&(x < 12)$ as an example: We know that the satisfying model has to meets the condition $z = 10$, so we can focus only on three possible intersections:

1. The intersection of $x = y + 1$ and $z = 10$
2. The intersection of $y = z + 1$ and $z = 10$
3. The intersection of $x = 12 - 1$ and $z = 10$

We only need to check $256 * 3$ cases in total, as compared to 256^3 case with the naive implementation.

4 3F-4 Equivalence

- **0 pts** Correct

Exercise 3F-3. Equivalence

Regular expression equivalence $e_1 \sim e_2$ is decidable, because a regular expression equivalence can be reduced to DFA (Deterministic Finite Automaton) equivalence and DFA equivalence is decidable.

We can translate a regular expression into an equivalent DFA by **first translating it into an equivalent NFA** (Non-deterministic Finite Automaton) with Thompson's construction, and **then to DFA** with subset construction algorithm. Now to show regular expression equivalence is decidable, we only need to show that FDA equivalence is decidable.

Let A and B be two arbitrary DFA. A and B are equivalent if and only if A and B accept the same strings up to length $m * n$, where m and n are the number of states in A and B. This is because if A and B are not equivalent, they must output different results for some string of length at most $m * n$. Since the number of strings of length up to $m * n$ is finite, we can solve this problem in finite amount of time. i.e. DFA equivalence is decidable.

Ref: https://cs.stackexchange.com/questions/92496/proving-that-dfa-equivalence-is-decidable

Exercise 3F-4. SAT Solving

The test 35 and 36 are slow because they have 3 variables while the rest of the test have at most 2 variables. This matters a lot because in the given code, the **arithmetic solver** works in a way that generates all possible combination of variables ranging from -127 to 128 and naively checks all possible combinations against conditions. This means for a input with n variables, the arithmetic solver takes $O(256^n)$ times to execute.

The performance could be made much better if we rewrite the arithmetic solver into a **linear programming solver**. Instead of sweeping through the whole variable space, we could focus on just the intersections of functions defined by linear inequalities. We could further improve by focusing on intersections that involves all the equalities. Take test case 26 $(x > y)\&\&(y > z)\&\&(z = 10)\&\&(x < 12)$ as an example: We know that the satisfying model has to meets the condition $z = 10$, so we can focus only on three possible intersections:

1. The intersection of $x = y + 1$ and $z = 10$
2. The intersection of $y = z + 1$ and $z = 10$
3. The intersection of $x = 12 - 1$ and $z = 10$

We only need to check $256 * 3$ cases in total, as compared to 256^3 case with the naive implementation.

5 3F-5 SAT Solving

- **0 pts** Correct