1 3F-1 Bookkeeping

- 0 pts Correct

Peer Review ID: 68499702 — enter this when you fill out your peer evaluation via gradescope

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

The large-step operational semantics rules of inference are listed below.

$e ::= e_1\ e_2$:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } s_1 \quad \vdash e_2 \text{ matches } s_1 \text{ leaving } s'}{\vdash e_1\ e_2 \text{ matches } s \text{ leaving } s'}$$

$e ::= e_1 \mid e_2$:

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

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

$e ::= e*$:

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

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

Exercise 3F-3. Regular Expression and Sets.

It is impossible to do the same thing for $e*$ or $e_1\ e_2$ because their rules of inference don't hvae a finite and fixed set of hypotheses.

For $e*$,

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

$$\frac{\vdash e \text{ matches } s \text{ leaving } S' \qquad \forall s_i \in S' \vdash e* \text{ matches } s_i \text{ leaving } S_i}{\vdash e* \text{ matches } s \text{ leaving } \cup_i S_i}$$

For $e_1\ e_2$,

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S' \qquad \forall s_i \in S' \vdash e_2 \text{ matches } s_i \text{ leaving } S_i}{\vdash e_1\ e_2 \text{ matches } s \text{ leaving } \cup_i S_i}$$

Since the size of S' can be arbitrarily large and can vary condition by condition, meaning that we don't have finite and fixed hypotheses. The two rules are not reasonable.

2 3F-2 Regular Expressions, Large Step

- 0 pts Correct

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

The large-step operational semantics rules of inference are listed below.

$e ::= e_1\ e_2$:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } s_1 \quad \vdash e_2 \text{ matches } s_1 \text{ leaving } s'}{\vdash e_1\ e_2 \text{ matches } s \text{ leaving } s'}$$

$e ::= e_1 \mid e_2$:

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

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

$e ::= e*$:

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

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

Exercise 3F-3. Regular Expression and Sets.

It is impossible to do the same thing for $e*$ or $e_1\ e_2$ because their rules of inference don't hvae a finite and fixed set of hypotheses.

For $e*$,

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

$$\frac{\vdash e \text{ matches } s \text{ leaving } S' \qquad \forall s_i \in S' \vdash e* \text{ matches } s_i \text{ leaving } S_i}{\vdash e* \text{ matches } s \text{ leaving } \cup_i S_i}$$

For $e_1\ e_2$,

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S' \qquad \forall s_i \in S' \vdash e_2 \text{ matches } s_i \text{ leaving } S_i}{\vdash e_1\ e_2 \text{ matches } s \text{ leaving } \cup_i S_i}$$

Since the size of S' can be arbitrarily large and can vary condition by condition, meaning that we don't have finite and fixed hypotheses. The two rules are not reasonable.

3 3F-3 Regular Expressions and Sets

- 0 pts Correct

Exercise 3F-4. Equivalence.

We assume that e_1 and e_2 operates on the same language. To compute whether 2 regular expressions e_1 and e_2 are equivalent, we need to

1. Construct NFAs for regular expressions (Algorithm 3.23). $e_1 \to N_1$ and $e_2 \to N_2$.

2. Convert NFAs to DFAs (Algorithm 3.20). $N_1 \to D_1$ and $N_2 \to D_2$.

3. Minimize the DFAs (Algorithm 3.39). $D_1 \to D_1'$ and $D_2 \to D_2'$.

4. Check whether the two minimized DFAs D_1' and D_2' are equivalent by DFS.

If D_1' is equivalent to D_2', e_1 and e_2 are equivalent; otherwise, e_1 and e_2 are not.

The algorithms mentioned above are from *Compilers: Principles, Techniques, and Tools*, 2nd edition.

Exercise 3F-5. SAT Solving.

The last two test cases (35 & 36) take longer because they involve more arithmetic inequalities than others.

To solve arithmetic inequalities, the function `Arith.arith` just goes over all 256 numbers for every variable to get the possible answer (line 68 - 71).

Listing 1: code snippet from `arith.ml`

```
61 (* for each variable, try all of values for it in a bounded range *)
62 let rec bounded_search variables model_sofar =
63   if StringSet.is_empty variables then
64     consider model_sofar
65   else begin
66     let variable = StringSet.choose variables in
67     let variables = StringSet.remove variable variables in
68     for i = lower_bound to upper_bound do
69       let model = StringMap.add variable i model_sofar in
70       bounded_search variables model
71     done
72   end
73 in
```

The brute-force way has a time complexity of $O(256^n)$ for n different arithmetic variables, which can be extremely time-consuming for larger n.

To improve the performance, we might use linear programming methods such as the Simplex Algorithm to solve the inequalities. That may give a better performance than the simple brute-force way.

4 3F-4 Equivalence

- 0 pts Correct

Exercise 3F-4. Equivalence.

We assume that e_1 and e_2 operates on the same language. To compute whether 2 regular expressions e_1 and e_2 are equivalent, we need to

1. Construct NFAs for regular expressions (Algorithm 3.23). $e_1 \to N_1$ and $e_2 \to N_2$.
2. Convert NFAs to DFAs (Algorithm 3.20). $N_1 \to D_1$ and $N_2 \to D_2$.
3. Minimize the DFAs (Algorithm 3.39). $D_1 \to D_1'$ and $D_2 \to D_2'$.
4. Check whether the two minimized DFAs D_1' and D_2' are equivalent by DFS.

If D_1' is equivalent to D_2', e_1 and e_2 are equivalent; otherwise, e_1 and e_2 are not.

The algorithms mentioned above are from *Compilers: Principles, Techniques, and Tools*, 2nd edition.

Exercise 3F-5. SAT Solving.

The last two test cases (35 & 36) take longer because they involve more arithmetic inequalities than others.

To solve arithmetic inequalities, the function `Arith.arith` just goes over all 256 numbers for every variable to get the possible answer (line 68 - 71).

Listing 1: code snippet from `arith.ml`

```
61 (* for each variable, try all of values for it in a bounded range *)
62 let rec bounded_search variables model_sofar =
63   if StringSet.is_empty variables then
64     consider model_sofar
65   else begin
66     let variable = StringSet.choose variables in
67     let variables = StringSet.remove variable variables in
68     for i = lower_bound to upper_bound do
69       let model = StringMap.add variable i model_sofar in
70       bounded_search variables model
71     done
72   end
73 in
```

The brute-force way has a time complexity of $O(256^n)$ for n different arithmetic variables, which can be extremely time-consuming for larger n.

To improve the performance, we might use linear programming methods such as the Simplex Algorithm to solve the inequalities. That may give a better performance than the simple brute-force way.

5 3F-5 SAT Solving

- 0 pts Correct

Peer Review ID: 68499702 — enter this when you fill out your peer evaluation via gradescope