1 3F-1 Bookkeeping

- **0 pts** Correct

Exercise 3F-2. Regular Expression, Large-Step [10 points]

Concatenation

$$\frac{\vdash e_1 \textbf{ matches } s \textbf{ leaving } s_1 \qquad \vdash e_2 \textbf{ matches } s_1 \textbf{ leaving } s_2}{\vdash e_1\ e_2 \textbf{ matches } s \textbf{ leaving } s_2}$$

Or

$$\frac{\vdash e_1 \textbf{ matches } s \textbf{ leaving } s_1}{\vdash e_1|e_2 \textbf{ matches } s \textbf{ leaving } s_1}$$

$$\frac{\vdash e_2 \textbf{ matches } s \textbf{ leaving } s_2}{\vdash e_1|e_2 \textbf{ matches } s \textbf{ leaving } s_2}$$

Kleene star

$$\frac{\vdash e_1 \textbf{ matches } s \textbf{ leaving } s_1 \qquad \vdash e_1* \textbf{ matches } s_1 \textbf{ leaving } s_2}{\vdash e_1* \textbf{ matches } s \textbf{ leaving } s_2}$$

$$\frac{}{\vdash e_1* \textbf{ matches } s \textbf{ leaving } s}$$

Exercise 3F-3. Regular Expression and Sets [5 points]

I claim that it is impossible to write operational semantics rules of inference given the restrictions that we have. This is because for the commands that are dependent on executing something first (concatenation and kleene star), the first command can return a set of size more than one, and the second command has no way to specify that we want to operate on every element within the initially returned set given the constraints.

Consider the following attempt to write a rule for Kleene star:

$$\frac{}{\vdash e_1* \textbf{ matches } s \textbf{ leaving } \{s\}}$$

$$\frac{\vdash e_1 \textbf{ matches } s \textbf{ leaving } S_1 = \{s_1 | s = e_1 :: s_1\} \qquad \vdash e_1* \textbf{ matches } s_1 \textbf{ leaving } S_2}{\vdash e_1* \textbf{ matches } s \textbf{ leaving } S_1 \cup S_2}$$

This doesn't work as S can have potentially more than one element depending on what the expression is. In order to return the correct output, we would have to run our recursive match command

2 3F-2 Regular Expressions, Large Step

- **0 pts** Correct

Exercise 3F-2. Regular Expression, Large-Step [10 points]

Concatenation

$$\frac{\vdash e_1 \textbf{ matches } s \textbf{ leaving } s_1 \qquad \vdash e_2 \textbf{ matches } s_1 \textbf{ leaving } s_2}{\vdash e_1\ e_2 \textbf{ matches } s \textbf{ leaving } s_2}$$

Or

$$\frac{\vdash e_1 \textbf{ matches } s \textbf{ leaving } s_1}{\vdash e_1|e_2 \textbf{ matches } s \textbf{ leaving } s_1}$$

$$\frac{\vdash e_2 \textbf{ matches } s \textbf{ leaving } s_2}{\vdash e_1|e_2 \textbf{ matches } s \textbf{ leaving } s_2}$$

Kleene star

$$\frac{\vdash e_1 \textbf{ matches } s \textbf{ leaving } s_1 \qquad \vdash e_1* \textbf{ matches } s_1 \textbf{ leaving } s_2}{\vdash e_1* \textbf{ matches } s \textbf{ leaving } s_2}$$

$$\frac{}{\vdash e_1* \textbf{ matches } s \textbf{ leaving } s}$$

Exercise 3F-3. Regular Expression and Sets [5 points]

I claim that it is impossible to write operational semantics rules of inference given the restrictions that we have. This is because for the commands that are dependent on executing something first (concatenation and kleene star), the first command can return a set of size more than one, and the second command has no way to specify that we want to operate on every element within the initially returned set given the constraints.

Consider the following attempt to write a rule for Kleene star:

$$\frac{}{\vdash e_1* \textbf{ matches } s \textbf{ leaving } \{s\}}$$

$$\frac{\vdash e_1 \textbf{ matches } s \textbf{ leaving } S_1 = \{s_1 | s = e_1 :: s_1\} \qquad \vdash e_1* \textbf{ matches } s_1 \textbf{ leaving } S_2}{\vdash e_1* \textbf{ matches } s \textbf{ leaving } S_1 \cup S_2}$$

This doesn't work as S can have potentially more than one element depending on what the expression is. In order to return the correct output, we would have to run our recursive match command

onall the elements that get returned. However, we can not specify that we want to do that without using a derivation inside a set constructor. Looking at the command provided, we can see that we are indeed only matching one of the potential elements in S.

Consider the following attmept to write a rule for concatenation:

$$\frac{\vdash e_1 \textbf{ matches } s \textbf{ leaving } \{s_1 | s_1 = e_1 :: s\} \qquad \vdash e_2 \textbf{ matches } s_1 \textbf{ leaving } S}{\vdash e_1\ e_2 \textbf{ matches } s \textbf{ leaving } S}$$

The same issue occurs. The second judgement has no way to specify all the elements of the first set given that we must used a finite and fixed amount of hypotheses and are disallowed judgements in the set construction. Consequently, if the expression generates a set of size more than one, our second judgement on the top has no way to handle it.

Exercise 3F-4. Equivalence [7 points]

I claim that $e_1 \sim e_2$ is decidable.

It is possible to decide an equivalent DFA for a regex expression, where the strings that the DFA accepts will be the exact ones that are in the set the regex leaves in the statement e **matches** s **leaving** S. It is possible to decide equivalences of DFAs by reducing each DFA to its minimal counterpart and then comparing them. Consequently, we can check the equivalence of two regex statements by checking the equivalence of the minimal DFAs corresponding to each regex statement.

Exercise 3C. SAT Solving

Submitted

Exercise 3F-5. SAT Solving [6 points]

When we're working pure propositional logic, all variables are independent, which makes analysis easier. On the other hand allowing theories introduces the possibility of having dependent variables. If we comb through the test cases, we can see that despite some of the other test cases using more than just boolean logic, only the last two test cases actually have dependent variables, which is what leads to the slowdown in runtime.

If we take a look at the arithmetic module provided to us, one of the first comments present is that it is "a very simple, but very inefficient, integer arithmetic constraint solver." Since this module is really what handles the introduction of theories, it makes sense that this module is what's causing the slowdown and is the most inefficient. Scanning through the module, we notice

3 3F-3 Regular Expressions and Sets

- **0 pts** Correct

onall the elements that get returned. However, we can not specify that we want to do that without using a derivation inside a set constructor. Looking at the command provided, we can see that we are indeed only matching one of the potential elements in S.

Consider the following attmept to write a rule for concatenation:

$$\frac{\vdash e_1 \textbf{ matches } s \textbf{ leaving } \{s_1 | s_1 = e_1 :: s\} \qquad \vdash e_2 \textbf{ matches } s_1 \textbf{ leaving } S}{\vdash e_1\ e_2 \textbf{ matches } s \textbf{ leaving } S}$$

The same issue occurs. The second judgement has no way to specify all the elements of the first set given that we must used a finite and fixed amount of hypotheses and are disallowed judgements in the set construction. Consequently, if the expression generates a set of size more than one, our second judgement on the top has no way to handle it.

Exercise 3F-4. Equivalence [7 points]

I claim that $e_1 \sim e_2$ is decidable.

It is possible to decide an equivalent DFA for a regex expression, where the strings that the DFA accepts will be the exact ones that are in the set the regex leaves in the statement e **matches** s **leaving** S. It is possible to decide equivalences of DFAs by reducing each DFA to its minimal counterpart and then comparing them. Consequently, we can check the equivalence of two regex statements by checking the equivalence of the minimal DFAs corresponding to each regex statement.

Exercise 3C. SAT Solving

Submitted

Exercise 3F-5. SAT Solving [6 points]

When we're working pure propositional logic, all variables are independent, which makes analysis easier. On the other hand allowing theories introduces the possibility of having dependent variables. If we comb through the test cases, we can see that despite some of the other test cases using more than just boolean logic, only the last two test cases actually have dependent variables, which is what leads to the slowdown in runtime.

If we take a look at the arithmetic module provided to us, one of the first comments present is that it is "a very simple, but very inefficient, integer arithmetic constraint solver." Since this module is really what handles the introduction of theories, it makes sense that this module is what's causing the slowdown and is the most inefficient. Scanning through the module, we notice

4 3F-4 Equivalence

- **0 pts** Correct

onall the elements that get returned. However, we can not specify that we want to do that without using a derivation inside a set constructor. Looking at the command provided, we can see that we are indeed only matching one of the potential elements in S.

Consider the following attmept to write a rule for concatenation:

$$\frac{\vdash e_1 \textbf{ matches } s \textbf{ leaving } \{s_1 | s_1 = e_1 :: s\} \qquad \vdash e_2 \textbf{ matches } s_1 \textbf{ leaving } S}{\vdash e_1\ e_2 \textbf{ matches } s \textbf{ leaving } S}$$

The same issue occurs. The second judgement has no way to specify all the elements of the first set given that we must used a finite and fixed amount of hypotheses and are disallowed judgements in the set construction. Consequently, if the expression generates a set of size more than one, our second judgement on the top has no way to handle it.

Exercise 3F-4. Equivalence [7 points]

I claim that $e_1 \sim e_2$ is decidable.

It is possible to decide an equivalent DFA for a regex expression, where the strings that the DFA accepts will be the exact ones that are in the set the regex leaves in the statement e **matches** s **leaving** S. It is possible to decide equivalences of DFAs by reducing each DFA to its minimal counterpart and then comparing them. Consequently, we can check the equivalence of two regex statements by checking the equivalence of the minimal DFAs corresponding to each regex statement.

Exercise 3C. SAT Solving

Submitted

Exercise 3F-5. SAT Solving [6 points]

When we're working pure propositional logic, all variables are independent, which makes analysis easier. On the other hand allowing theories introduces the possibility of having dependent variables. If we comb through the test cases, we can see that despite some of the other test cases using more than just boolean logic, only the last two test cases actually have dependent variables, which is what leads to the slowdown in runtime.

If we take a look at the arithmetic module provided to us, one of the first comments present is that it is "a very simple, but very inefficient, integer arithmetic constraint solver." Since this module is really what handles the introduction of theories, it makes sense that this module is what's causing the slowdown and is the most inefficient. Scanning through the module, we notice

that we effectively consider all possible values (in a given range) for all possible variables. However, this means that we're also considering values that we can guarantee aren't possible. For example, if we were given $x > y$, this module would still check $x = 10, y = 100$. The first fix I would make is to add some logic constraining the search range of dependent variables in the cases that the dependency must be true. If, for example a statement looked like $x > y||y >= x$, then there is not much reason to constrain the search range for our variables. Furthermore, we could additionally introduce some sort of backtracking algorithm to more intelligently attempt values that fit these dependency constraints.

5 3F-5 SAT Solving

- **0 pts** Correct