1 3F-1 Bookkeeping

- 0 pts Correct

Peer Review ID: 68512903 — enter this when you fill out your peer evaluation via gradescope

2F-2

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } \mathsf{s}_1 \qquad \vdash e_2 \text{ matches } s_1 \text{ leaving } s_2}{\vdash e_1\ e_2 \text{ matches } s \text{ leaving } s_2}$$

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

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

2F-3

This is not possible with the given framework. Applying $e*$ involves applying e a number of times that is not fixed and finite. Large-step semantics represent repeated application such as this by recursion, but that would involve recursively matching $e*$ against all strings in some some set S left by matching e, which still leaves a number of hypothesis that is not fixed and finite.

2F-4

A regular expression as we have defined it can be expressed as an non-deterministic finite automaton (NFA), as both regular expressions and NFAs correspond to regular languages. NFAs are in turn equivalent to deterministic finite automatons (DFAs).

For any DFA, there exists a unique minimal DFA that matches the same languages. Therefore, equivalence of regular expressions can be computed generally by converting to a DFA, then finding the minimal DFA, the comparing the results. This is a very inefficient process, but it is decidable.

2F-5

The last two included tests take a long time due to the number of variables and absolute value used in the comparison. This means that it must find specific values (or in the first case check all possible values) within the bounded search. There are also three numeric variables in this expression, which quickly gets out of hand with the bounded search. I would rewrite the arithmetic module to be smarter about choosing or eliminating potential values rather than just searching the whole space for every variable.

The easiest point to improve would be using the values for equality expressions involving a variable and a number expected to evaluate to true as initial assignments.

2 3F-2 Regular Expressions, Large Step

- 0 pts Correct

2F-2

$$\frac{\vdash e_1 \textsf{ matches } s \textsf{ leaving } \textsf{s}_1 \qquad \vdash e_2 \textsf{ matches } s_1 \textsf{ leaving } s_2}{\vdash e_1\ e_2 \textsf{ matches } s \textsf{ leaving } s_2}$$

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

$$\frac{}{\vdash e* \textsf{ matches } s \textsf{ leaving } s} \qquad \frac{\vdash e \textsf{ matches } s \textsf{ leaving } s_1 \qquad \vdash e* \textsf{ matches } s_1 \textsf{ leaving } s_2}{\vdash e* \textsf{ matches } s \textsf{ leaving } s_2}$$

2F-3

This is not possible with the given framework. Applying $e*$ involves applying e a number of times that is not fixed and finite. Large-step semantics represent repeated application such as this by recursion, but that would involve recursively matching $e*$ against all strings in some some set S left by matching e, which still leaves a number of hypothesis that is not fixed and finite.

2F-4

A regular expression as we have defined it can be expressed as an non-deterministic finite automaton (NFA), as both regular expressions and NFAs correspond to regular languages. NFAs are in turn equivalent to deterministic finite automatons (DFAs).

For any DFA, there exists a unique minimal DFA that matches the same languages. Therefore, equivalence of regular expressions can be computed generally by converting to a DFA, then finding the minimal DFA, the comparing the results. This is a very inefficient process, but it is decidable.

2F-5

The last two included tests take a long time due to the number of variables and absolute value used in the comparison. This means that it must find specific values (or in the first case check all possible values) within the bounded search. There are also three numeric variables in this expression, which quickly gets out of hand with the bounded search. I would rewrite the arithmetic module to be smarter about choosing or eliminating potential values rather than just searching the whole space for every variable.

The easiest point to improve would be using the values for equality expressions involving a variable and a number expected to evaluate to true as initial assignments.

3 3F-3 Regular Expressions and Sets

- 0 pts Correct

2F-2

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } \mathsf{s}_1 \qquad \vdash e_2 \text{ matches } s_1 \text{ leaving } s_2}{\vdash e_1\ e_2 \text{ matches } s \text{ leaving } s_2}$$

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

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

2F-3

This is not possible with the given framework. Applying $e*$ involves applying e a number of times that is not fixed and finite. Large-step semantics represent repeated application such as this by recursion, but that would involve recursively matching $e*$ against all strings in some some set S left by matching e, which still leaves a number of hypothesis that is not fixed and finite.

2F-4

A regular expression as we have defined it can be expressed as an non-deterministic finite automaton (NFA), as both regular expressions and NFAs correspond to regular languages. NFAs are in turn equivalent to deterministic finite automatons (DFAs).

For any DFA, there exists a unique minimal DFA that matches the same languages. Therefore, equivalence of regular expressions can be computed generally by converting to a DFA, then finding the minimal DFA, the comparing the results. This is a very inefficient process, but it is decidable.

2F-5

The last two included tests take a long time due to the number of variables and absolute value used in the comparison. This means that it must find specific values (or in the first case check all possible values) within the bounded search. There are also three numeric variables in this expression, which quickly gets out of hand with the bounded search. I would rewrite the arithmetic module to be smarter about choosing or eliminating potential values rather than just searching the whole space for every variable.

The easiest point to improve would be using the values for equality expressions involving a variable and a number expected to evaluate to true as initial assignments.

4 3F-4 Equivalence

- 0 pts Correct

2F-2

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

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

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

2F-3

This is not possible with the given framework. Applying $e*$ involves applying e a number of times that is not fixed and finite. Large-step semantics represent repeated application such as this by recursion, but that would involve recursively matching $e*$ against all strings in some some set S left by matching e, which still leaves a number of hypothesis that is not fixed and finite.

2F-4

A regular expression as we have defined it can be expressed as an non-deterministic finite automaton (NFA), as both regular expressions and NFAs correspond to regular languages. NFAs are in turn equivalent to deterministic finite automatons (DFAs).

For any DFA, there exists a unique minimal DFA that matches the same languages. Therefore, equivalence of regular expressions can be computed generally by converting to a DFA, then finding the minimal DFA, the comparing the results. This is a very inefficient process, but it is decidable.

2F-5

The last two included tests take a long time due to the number of variables and absolute value used in the comparison. This means that it must find specific values (or in the first case check all possible values) within the bounded search. There are also three numeric variables in this expression, which quickly gets out of hand with the bounded search. I would rewrite the arithmetic module to be smarter about choosing or eliminating potential values rather than just searching the whole space for every variable.

The easiest point to improve would be using the values for equality expressions involving a variable and a number expected to evaluate to true as initial assignments.

5 3F-5 SAT Solving

- 0 pts Correct

Peer Review ID: 68512903 — enter this when you fill out your peer evaluation via gradescope