1 3F-1 Bookkeeping

- 0 pts Correct

Peer Review ID: 68553381 — enter this when you fill out your peer evaluation via gradescope

Exercise 3F-2.

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } s' \quad \vdash e_2 \text{ matches } s' \text{ leaving } s''}{\vdash e_1e_2 \text{ matches } s \text{ leaving } s''}$$

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

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

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

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

Exercise 3F-3.

It cannot be done, with the given restrictions on rule constructions.

For example, this is an attempt at making a rule of inference for e_1e_2:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S' \quad \vdash e_2 \text{ matches } S' \text{ leaving } S???}{\vdash e_1e_2 \text{ matches } s \text{ leaving } S}$$

As seen in the $\vdash (\text{"}h\text{"} \mid \text{"}e\text{"})*$ matches "*hello*" leaving example in Question 2, a given regex may derive multiple instances of suffices. Hence we need to portray all of e_1's possible suffices as a set if we want our semantics to be deterministic. However, that would mean e_2 will have to consider all those possible suffices (portrayed as S' above), which cannot be done with the given restrictions concerning there being finite and fixed set of hypotheses.

Another attempt at making a rule for e_1e_2:

$$\frac{s = e_1 :: e_2 :: S???}{\vdash e_1e_2 \text{ matches } s \text{ leaving } S}$$

This attempt shows that for all possible suffices S, a corresponding string s can be constructed by appending the suffix S to the regexes e_1 and e_2. However, this is not a correct construction of the string s, since e_1 and e_2 are both regular expressions, not strings, and S is a set of strings, not a string. Hence the constructed s is not a valid string In this scenario.

2 3F-2 Regular Expressions, Large Step

- 0 pts Correct

Peer Review ID: 68553381 — enter this when you fill out your peer evaluation via gradescope

Exercise 3F-2.

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

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

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

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

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

Exercise 3F-3.

It cannot be done, with the given restrictions on rule constructions.

For example, this is an attempt at making a rule of inference for $e_1 e_2$:

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

As seen in the $\vdash$ $(''h'' \mid ''e'') *$ matches $''hello''$ leaving example in Question 2, a given regex may derive multiple instances of suffices. Hence we need to portray all of e_1's possible suffices as a set if we want our semantics to be deterministic. However, that would mean e_2 will have to consider all those possible suffices (portrayed as S' above), which cannot be done with the given restrictions concerning there being finite and fixed set of hypotheses.

Another attempt at making a rule for $e_1 e_2$:

$$\frac{s = e_1 :: e_2 :: S???}{\vdash e_1 e_2 \textsf{ matches } s \textsf{ leaving } S}$$

This attempt shows that for all possible suffices S, a corresponding string s can be constructed by appending the suffix S to the regexes e_1 and e_2. However, this is not a correct construction of the string s, since e_1 and e_2 are both regular expressions, not strings, and S is a set of strings, not a string. Hence the constructed s is not a valid string In this scenario.

3 3F-3 Regular Expressions and Sets

- 0 pts Correct

Peer Review ID: 68553381 — enter this when you fill out your peer evaluation via gradescope

Exercise 3F-4.

The equivalence of regular expressions is decideable. All regexes can be turned into a Deterministic Finite Automaton (DFA), and all equivalent regexes have the same unique minimal DFA. Hence all that is needed to do is turn the regexes into DFA, minimalize them, and compare. If they are equal, the regexes are equivalent.

Exercise 3F-5.

The current implementation of the program makes the arithmetic solver replace each arithmetic variable given from the SAT solver sequentially from -127 to 128 and evaluate the resulting statements one-by-one, case-by-case. In case of the two last test cases (test-35.input and test-36.input), there are three integer variables x, y, and z. (The other cases have at most two.) This basically means for these last two cases, the solver will go through three nested for loops, exchanging integer values from -127 to 128 for each of the variables, until it finds a solution or it runs out of possiblities. This is why these two take so long - it takes cubic time to complete. (In fact test-35.input has no solution, which means it will interate through all the possible combos before finishing, meaning it takes even longer) Therefore, if I had to change one module to improve performance, I would change the arith module. As for how, one could add a simple semantic "sensor" that can read the integer inequalities, and iterate only through the possible cases and ignore the impossible ones (for example, since test-35.input contains $x < 12$ as one of the predicates, I will not bother computing after the loop passes $x = 12$ and up.)

The "egregious defect" seems to be situated in exp_to_cnf in exp.ml. While I won't try to elaborate on the underlying reasons why it happens, it appears that when trying to convert some of the test cases (i.e. test-27.input), it is unable to group the same predicates (e.g. $(x) = (5)$), and as a result, it creates multiple instances of the same predicate under different names.

4 3F-4 Equivalence

- 0 pts Correct

Exercise 3F-4.

The equivalence of regular expressions is decideable. All regexes can be turned into a Deterministic Finite Automaton (DFA), and all equivalent regexes have the same unique minimal DFA. Hence all that is needed to do is turn the regexes into DFA, minimalize them, and compare. If they are equal, the regexes are equivalent.

Exercise 3F-5.

The current implementation of the program makes the arithmetic solver replace each arithmetic variable given from the SAT solver sequentially from -127 to 128 and evaluate the resulting statements one-by-one, case-by-case. In case of the two last test cases (test-35.input and test-36.input), there are three integer variables x, y, and z. (The other cases have at most two.) This basically means for these last two cases, the solver will go through three nested for loops, exchanging integer values from -127 to 128 for each of the variables, until it finds a solution or it runs out of possiblities. This is why these two take so long - it takes cubic time to complete. (In fact test-35.input has no solution, which means it will interate through all the possible combos before finishing, meaning it takes even longer) Therefore, if I had to change one module to improve performance, I would change the arith module. As for how, one could add a simple semantic "sensor" that can read the integer inequalities, and iterate only through the possible cases and ignore the impossible ones (for example, since test-35.input contains $x < 12$ as one of the predicates, I will not bother computing after the loop passes $x = 12$ and up.)

The "egregious defect" seems to be situated in exp_to_cnf in exp.ml. While I won't try to elaborate on the underlying reasons why it happens, it appears that when trying to convert some of the test cases (i.e. test-27.input), it is unable to group the same predicates (e.g. $(x) = (5)$), and as a result, it creates multiple instances of the same predicate under different names.

5 3F-5 SAT Solving

- 0 pts Correct