1 3F-1 Bookkeeping

- **0 pts** Correct

All subsequent answers should appear after the first page of your submission and may be shared publicly during peer review.

Exercise 3F-2. Regular Expression, Large-Step [10 points]. Regular Expressions are commonly used as abstractions for string matching. Here is an abstract grammar for regular expressions:

e	::=	"x"	singleton — matches the character $\hat{x}$
	|	empty	skip — matches the empty string
	|	$e_1\ e_2$	concatenation — matches e_1 followed by e_2
	|	$e_1 \mid e_2$	or — matches e_1 or e_2
	|	$e*$	Kleene star — matches 0 or more occurrence of e
	|	.	matches any single character
	|	["x" − "y"]	matches any character between $\hat{x}$ and $\hat{y}$ inclusive
	|	$e+$	matches 1 or more occurrences of e
	|	$e?$	matches 0 or 1 occurrence of e

We will call the first five cases the *primary* forms of regular expressions. The last four cases can be defined in terms of the first five. We also give an abstract grammar for strings (modeled as lists of characters):

s	::=	nil	empty string
	|	"x" :: s	string with first character $\hat{x}$ and other characters s

We write "bye" as shorthand for "b" :: "y" :: "e" :: nil. This exercise requires you to give large-step operational semantics rules of inference related to regular expressions matching strings. We introduce a judgment:

$$\vdash e \text{ matches } s \text{ leaving } s'$$

The interpretation of the judgment is that the regular expression e matches some prefix of the string s, leaving the suffix s' unmatched. If $s' = \text{nil}$ then r matched s exactly. Examples:

$$\vdash \text{"h"}(\text{"e"}+) \text{ matches "hello" leaving "llo"}$$

Note that this operational semantics may be considered *non-deterministic* because we expect to be able to derive all three of the following:

$$\begin{array}{l} \vdash (\text{"h"} \mid \text{"e"})* \text{ matches "hello" leaving "ello"} \\ \vdash (\text{"h"} \mid \text{"e"})* \text{ matches "hello" leaving "hello"} \\ \vdash (\text{"h"} \mid \text{"e"})* \text{ matches "hello" leaving "llo"} \end{array}$$

Here are two rules of inference:

$$\frac{s = \text{"x"} :: s'}{\vdash \text{"x" matches } s \text{ leaving } s'} \qquad \frac{}{\vdash \text{empty matches } s \text{ leaving } s}$$

Give large-step operational semantics rules of inference for the other three primal regular expressions.

We start with the concatenation rule. This one is fairly straightforward, and is much like the rule we saw for commands.

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

For or, we simply need two rules, one for each regular expression.

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

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

Lastly, we are left with the Kleene star. We need a rule for matching 0 times, where the original string will be left.

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

Second, we can define a recursive rule to represent matching one or more times. This works because due to our first rule, s'' could be equal to s', and the expression terminates there.

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

Exercise 3F-3. Regular Expression and Sets [5 points]. We want to update our operational semantics for regular expressions to capture multiple suffices. We want our new operational semantics to be deterministic — it return the set of all possible answers from the single-answer operational semantics above. We introduce a new judgment:

$$\vdash e \ \mathsf{matches}\ s \ \mathsf{leaving}\ S$$

And use rules of inference like the following:

$$\frac{}{\vdash \text{"x"} \ \mathsf{matches}\ s \ \mathsf{leaving}\ \{s' \mid s = \text{"x"} :: s'\}} \qquad \frac{}{\vdash \mathsf{empty}\ \mathsf{matches}\ s \ \mathsf{leaving}\ \{s\}}$$

$$\frac{\vdash e_1 \ \mathsf{matches}\ s \ \mathsf{leaving}\ S \quad \vdash e_2 \ \mathsf{matches}\ s \ \mathsf{leaving}\ S'}{\vdash e_1 \mid e_2 \ \mathsf{matches}\ s \ \mathsf{leaving}\ S \cup S'}$$

You must do one of the following:

- *either* give operational semantics rules of inference for $e*$ and $e_1 e_2$. You may *not* place a derivation inside a set constructor, as in: $\{x \mid \exists y.\ \vdash e \ \mathsf{matches}\ x \ \mathsf{leaving}\ y\}$. Each inference rules must have a finite and fixed set of hypotheses.

2 3F-2 Regular Expressions, Large Step

- **0 pts** Correct

Give large-step operational semantics rules of inference for the other three primal regular expressions.

We start with the concatenation rule. This one is fairly straightforward, and is much like the rule we saw for commands.

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

For or, we simply need two rules, one for each regular expression.

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

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

Lastly, we are left with the Kleene star. We need a rule for matching 0 times, where the original string will be left.

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

Second, we can define a recursive rule to represent matching one or more times. This works because due to our first rule, s'' could be equal to s', and the expression terminates there.

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

Exercise 3F-3. Regular Expression and Sets [5 points]. We want to update our operational semantics for regular expressions to capture multiple suffices. We want our new operational semantics to be deterministic — it return the set of all possible answers from the single-answer operational semantics above. We introduce a new judgment:

$$\vdash e \ \mathsf{matches}\ s \ \mathsf{leaving}\ S$$

And use rules of inference like the following:

$$\frac{}{\vdash \text{"x"} \ \mathsf{matches}\ s \ \mathsf{leaving}\ \{s' \mid s = \text{"x"} :: s'\}} \qquad \frac{}{\vdash \mathsf{empty}\ \mathsf{matches}\ s \ \mathsf{leaving}\ \{s\}}$$

$$\frac{\vdash e_1 \ \mathsf{matches}\ s \ \mathsf{leaving}\ S \quad \vdash e_2 \ \mathsf{matches}\ s \ \mathsf{leaving}\ S'}{\vdash e_1 \mid e_2 \ \mathsf{matches}\ s \ \mathsf{leaving}\ S \cup S'}$$

You must do one of the following:

- *either* give operational semantics rules of inference for $e*$ and $e_1 e_2$. You may *not* place a derivation inside a set constructor, as in: $\{x \mid \exists y.\ \vdash e \ \mathsf{matches}\ x \ \mathsf{leaving}\ y\}$. Each inference rules must have a finite and fixed set of hypotheses.

- *or* argue in one or two sentences that it cannot be done correctly in the given framework. Back up your argument by presenting two attempted but "wrong" rules of inference and show that each one is either unsound or incomplete with respect to our intuitive notion of regular expression matching.

Part of doing research is getting stuck. When you get stuck, you must be able to recognize whether "you are just missing something" or "the problem is actually impossible".

The constraint of not being able to put derivations inside of set constructors as well as the unbounded nature of the Kleene star suggests that we are not able to write complete rules in the given framework. A naive non-trivial Kleene star rule could look like the following:

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

However this is clearly unsound, as regexes are not defined to work on sets of strings.

This could be "fixed" by trying to match individual strings from the set. This would be that rule for concatenation:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S \qquad \vdash e_2 \text{ matches } x, \forall x \in S \text{ leaving } S'}{\vdash e_1 e_2 \text{ matches } s \text{ leaving } S \cup \{ \bigcup_{s_i \in S'} s_i \}}$$

However, this is similarly unsound because we don't know the size of the set S or the set S'.

Exercise 3F-4. Equivalence [7 points]. In the class notes (usually marked as "optional material" for the lecture component of the class but relevant for this question) we defined an equivalence relation $c_1 \sim c_2$ for IMP commands. Computing equivalence turned out to be undecideable: $c \sim c$ iff c halts. We can define a similar equivalence relation for regular expressions: $e_1 \sim e_2$ iff $\forall s \in S.\ \vdash e_1 \textsf{ matches } s \textsf{ leaving } S_1 \ \wedge \ \vdash e_2 \textsf{ matches } s \textsf{ leaving } S_2 \implies S_1 = S_2$ (note that we are using an "updated" operational semantics that returns the set of all possible matched suffices, as in the previous problem).

You must *either* claim that $e_1 \sim e_2$ is undecideable by reducing it to the halting problem *or* explain in two or three sentences how to compute it. You may assume that I the reader is familiar with the relevant literature.

Interestingly, this problem actually is decidable. We can convert any regex to an equivalent discrete finite automaton. Given two DFAs corresponding to two regular expressions, we can minimize these DFAs. Now, we can compare the states in these minimal DFA's to see if they are all equivalent. If they are, the regular expressions must also be equivalent.

3 3F-3 Regular Expressions and Sets

- **0 pts** Correct

- *or* argue in one or two sentences that it cannot be done correctly in the given framework. Back up your argument by presenting two attempted but "wrong" rules of inference and show that each one is either unsound or incomplete with respect to our intuitive notion of regular expression matching.

Part of doing research is getting stuck. When you get stuck, you must be able to recognize whether "you are just missing something" or "the problem is actually impossible".

The constraint of not being able to put derivations inside of set constructors as well as the unbounded nature of the Kleene star suggests that we are not able to write complete rules in the given framework. A naive non-trivial Kleene star rule could look like the following:

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

However this is clearly unsound, as regexes are not defined to work on sets of strings.

This could be "fixed" by trying to match individual strings from the set. This would be that rule for concatenation:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S \qquad \vdash e_2 \text{ matches } x, \forall x \in S \text{ leaving } S'}{\vdash e_1 e_2 \text{ matches } s \text{ leaving } S \cup \{ \bigcup_{s_i \in S'} s_i \}}$$

However, this is similarly unsound because we don't know the size of the set S or the set S'.

Exercise 3F-4. Equivalence [7 points]. In the class notes (usually marked as "optional material" for the lecture component of the class but relevant for this question) we defined an equivalence relation $c_1 \sim c_2$ for IMP commands. Computing equivalence turned out to be undecideable: $c \sim c$ iff c halts. We can define a similar equivalence relation for regular expressions: $e_1 \sim e_2$ iff $\forall s \in S.\ \vdash e_1 \textsf{ matches } s \textsf{ leaving } S_1 \ \wedge \ \vdash e_2 \textsf{ matches } s \textsf{ leaving } S_2 \implies S_1 = S_2$ (note that we are using an "updated" operational semantics that returns the set of all possible matched suffices, as in the previous problem).

You must *either* claim that $e_1 \sim e_2$ is undecideable by reducing it to the halting problem *or* explain in two or three sentences how to compute it. You may assume that I the reader is familiar with the relevant literature.

Interestingly, this problem actually is decidable. We can convert any regex to an equivalent discrete finite automaton. Given two DFAs corresponding to two regular expressions, we can minimize these DFAs. Now, we can compare the states in these minimal DFA's to see if they are all equivalent. If they are, the regular expressions must also be equivalent.

4 3F-4 Equivalence

- **0 pts** Correct

Exercise 3C. SAT Solving. Download the Homework 3 code pack from the course web page. Update the skeletal SMT solver so that it correctly integrates the given DPLL-style CNF SAT solver with the given theory of bounded arithmetic. In particular, you must update only the Main.solve function. Your updated solver must be correct. This notably implies that it must correctly handle all of the included test cases — we use `diff` for some testing, but if you change only the listed method you should end up with the same answers as the reference.

In addition, create an example "tricky" input that can be parsed by our test harness.

Submit your `.ml` and `.input` files.

Exercise 3F-5. SAT Solving [6 points]. Why do the last two included tests take such a comparatively long time? Impress me with your knowledge of DPLL(T) — feel free to use information from the assigned reading or related papers, not just from the lecture slides. I am looking for a reasonably detailed answer. Include a discussion of which single module you would rewrite first to improve performance, as well as how you would change that module.

Potential bonus point: The provided code contains at least one fairly egregious defect. Comment.

The last two tests take comparatively longer because they contain a variable (y) that is only bound to constraints with other variables, no actual hard numbers. This is significant because DPLL(T) unit propagation can create cycles in the implication graph, and this will necessarily increase the search time.

The egregious defect in the code is that the arithmetic solver is doing an exhaustive search, which probably isn't necessary.

Submission. Turn in the formal component of the assignment as a single PDF document via the `gradescope` website. Your name and Michigan email address must appear on the first page of your PDF submission but may not appear anywhere else. Turn in the coding component of the assignment via the `autograder.io` website.

5 3F-5 SAT Solving

- **0 pts** Correct