1 3F-1 Bookkeeping

- **0 pts** Correct

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.

Solution $e_1\ e_2$:

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

$e_1 \mid e_2$:

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

$e*$:

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

2 3F-2 Regular Expressions, Large Step

- **0 pts** Correct

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:

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

And use rules of inference like the following:

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

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

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: $\{x \mid \exists y.\ \vdash e \text{ matches } x \text{ leaving } y\}$. 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".

Solution With the current framework, it is not possible to create operational semantics rules of inferences for $e*$ and $e_1\ e_2$. This is because evaluating $e*$ and $e_1\ e_2$ involve evaluating multiple regular expressions in sequence, which cannot be captured by rules with a finite and fixed set of hypotheses now that the regular expressions leave sets of multiple suffices. Consider the following incorrect rules of inference:

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

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S_1 \quad \vdash e_2 \text{ matches } s \text{ leaving } S_2}{\vdash e_1\ e_2 \text{ matches } s \text{ leaving } S_1 \cup S_2}$$

These two rules handle the issue of the set of suffices of e_1 being unknown in two ways. The top rule only works when e_1 leaves a singleton, but this rule is incomplete because it cannot be used to prove $\vDash$ ("h" | "i")? ("h" | "i") matches "hi" leaving {"i", nil} because the first expression leaves multiple suffices. The bottom rule tries to work with the fixed s, but is unsound because it can be used to prove the untrue statement "h" "h" matches "hi" leaving {"i"}

3 3F-3 Regular Expressions and Sets

- **0 pts** Correct

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 $\forall s \in S.\ \vdash e_1 \ \mathsf{matches}\ s\ \mathsf{leaving}\ S_1 \ \wedge\ \vdash e_2\ \mathsf{matches}\ s\ \mathsf{leaving}\ S_2 \implies S_1 = S_2$ (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.

Solution It is possible to to determine if two regular expressions are equivalent. Unlike programs, every finite regular expression will halt when evaluated on a finite input string because the regular expression evaluation can be described as a finite state machine that reads through the input string from beginning to end. To determine whether two regular expressions are equivalent, we can compare the finite state machines that are used to evaluate them.

4 3F-4 Equivalence

- **0 pts** Correct

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.

Solution The last two test cases involve a conjunction of four arithmetic clauses and no Boolean clauses. The bulk of the work for determining if the proposition is satisfiable falls to a single call to the arithmetic theorem solver. Inspecting how the arithmetic solver works reveals that its complexity is exponential with respect to the number of variables, explaining why the last two test cases, which contain four variables, take much longer than the previous test cases, which contain at most two variables.

To determine if the given arithmetic clauses are satisfiable, the arithmetic solver tries all possible combinations of values ranging from -127 to 128 for all the given variables to see if any combination satisfies the given clauses. The number of possible combinations is thus exponential with respect to the number of variables. This also explains why the second-to-last test case takes longer than the last test case even though they contain the same number of variables, as the second-to-last test is unsatisfiable, which the solver only discovers after completing the exhaustive search, while the last test case is satisfiable, allowing the solver to terminate early.

Other test cases have more complicated Boolean expressions that rely on the DPLL SAT solver, but the use of heuristics and a smaller search space makes solving those propositions faster.

The bounded search space used by the provided arithmetic solver also means that the solver will report some propositions as unsatisfiable even though the proposition is mathematically and logically satisfiable. For instance, the solver will report "$x < -200$" as unsatisfiable.

5 3F-5 SAT Solving

- **0 pts** Correct