No questions assigned to the following page.

Exercise 3F-1. 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.

Peer Review ID: 306712878 — enter this when you fill out your peer evaluation via gradescope

Questions assigned to the following page: 2 and 3

Answer

$$\frac{\vdash \mathsf{e}_1 \text{ matches } s \text{ leaving } s'' \qquad \vdash \mathsf{e}_2 \text{ matches } s'' \text{ leaving } s'}{\vdash \mathsf{e}_1\ \mathsf{e}_2 \text{ matches } s \text{ leaving } s'} \qquad \frac{\vdash \mathsf{e}_1 \text{ matches } s \text{ leaving } s'}{\vdash \mathsf{e}_1 \mid \mathsf{e}_2 \text{ matches } s \text{ leaving } s'}$$

$$\frac{\vdash \mathsf{e}_2 \text{ matches } s \text{ leaving } s'}{\vdash \mathsf{e}_1 \mid \mathsf{e}_2 \text{ matches } s \text{ leaving } s'} \qquad \frac{}{\vdash \mathsf{e}* \text{ matches } s \text{ leaving } s}$$

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

Exercise 3F-2. 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_1e_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".

Answer It cannot be done correctly in the given framwork. For example, we try to write the rules of e_1e_2. The correct one should be

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S_1 \quad \forall s' \in S_1, \vdash e_2 \text{ matches } s' \text{ leaving } S_2}{\vdash \mathsf{e}_1\ \mathsf{e}_2 \text{ matches } s \text{ leaving } \bigcup_{s' \in S_1} S_2}$$

Questions assigned to the following page: 4 and 3

but this rule is not allowed because its second "premise" is not a fixed finite list of hypotheses. It "unrolls" to one hypothesis for each $s' \in S_1$.
We use $e*$ for another example, the correct one should be

$$\frac{\vdash e \text{ matches } s \text{ leaving } S_1 \quad \forall s' \in S_1, \vdash e_2 \text{ matches } s' \text{ leaving } S_2}{\vdash \mathsf{e*} \text{ matches } s \text{ leaving } \{s\} \cup \bigcup_{s' \in S_1} S_2}$$

which again is not permitted because the number of premises depends on the (arbitrary) size of S_1.

Exercise 3F-3. 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 $\forall s \in S.\ \vdash e_1 \text{ matches } s \text{ leaving } S_1 \wedge \vdash e_2 \text{ matches } s \text{ leaving } S_2 \implies S_1 = S_2$ (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.

Answer $e_1 \sim e_2$ is undecideable. Let's treat pair (e_1,e_2) as an input of program s. e_1 e_2 are regular expressions and program s will only halt if S_1 (the remainder of applying e_1 to a string) and S_2 (the remainder of applying e_2 to the same string) are the same. Then, $e_1 \sim e_2$ iff program s will halt for all strings. However, accroding to the halting problem, we know it is impossible. So, $e_1 \sim e_2$ is undecideable.

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-4. 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.

Question assigned to the following page: 5

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

Answer The tests run slowly because the arithmetic solver uses an exhaustive search over a large bounded domain, and the integration with the SAT solver does little to prune the search space until after a full Boolean model is produced. For unsatisfiable cases, the solver may examine nearly the entire search space; for satisfiable cases, many candidates may be tried before the right one is found.
The arith.ml module is the most critical for performance improvements. Rewriting it to use constraint propagation techniques, interval reasoning, and incremental conflict analysis (instead of brute-force enumeration) would yield the most benefit.
The coarse conflict clause learning, where the conflict clause simply negates all theory literals in the current Boolean model, is a significant defect. It over-prunes the search space and forces repeated, inefficient re-computation. A more precise conflict explanation mechanism should be implemented.

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.