Question assigned to the following page: 2

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 $\widehat{x}$
	|	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 $\widehat{x}$ and $\widehat{y}$ 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 $\widehat{x}$ 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:

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

Peer Review ID: 311035788 — enter this when you fill out your peer evaluation via gradescope

Questions assigned to the following page: 3 and 2

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

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

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

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

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".

Solution: This can not be done correctly in the given framework. In our operational semantics, the suffix of strings can be viewed as states. I think the process of converting nondeterministic derivation rules to deterministic ones would be similar in spirit to the conversion from NFA to DFA. The deterministic states would be subset of states (strings). The given judgment should be updated to $\vdash e \text{ matches } S \text{ leaving } S'$, where S and S' are sets

Questions assigned to the following page: 3 and 4

of strings. The current framework is likely incomplete since the given judgment effectively matches only singleton sets $\{s\}$.

My attempt for the rule of $e_1 e_2$ is:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } \{s_1\} \quad \vdash e_2 \text{ matches } s_1 \text{ leaving } S}{\vdash e_1 e_2 \text{ matches } s \text{ leaving } S'}$$

Since the set of hypotheses have to be fixed, I cannot write the following as hypotheses of the rule.

$$\vdash e_1 \text{ matches } s \text{ leaving } S'$$
$$\vdash e_2 \text{ matches } s_1 \text{ leaving } S,\ \forall s_1 \in S'$$

My attempt for the rule of $e*$ is:

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

$$\frac{\vdash e \text{ matches } s \text{ leaving } \{s_1\} \quad \vdash e* \text{ matches } s_1 \text{ leaving } S'}{\vdash e* \text{ matches } s \text{ leaving } S'}$$

The rules are incomplete since they restrict the first hypothesis to leave with singleton sets.

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.

Solution: The problem is decidable. Given e_1, e_2, use the Thompson's Construction to derive the DFAs D_1 and D_2 that accepts e_1 and e_2, respectively. Construct a new DFA D that accepts the difference of $\mathcal{L}(D_1)$ and $\mathcal{L}(D_2)$. This DFA D can be constructed by taking the product of D_1 and D_2. The accepting states for D are the product states (s_1, s_2) such that exactly one of s_1, s_2 is originally an accepting state of D_1 or D_2. Finally, check the emptiness of the language $\mathcal{L}(D)$ accepted by D. The language $\mathcal{L}(D)$ is empty if and only if the accepting states of D are not reachable. If $\mathcal{L}(D)$ is empty, we have $e_1 \sim e_2$; otherwise we have $e_1 \not\sim e_2$.

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

Question assigned to the following page: 5

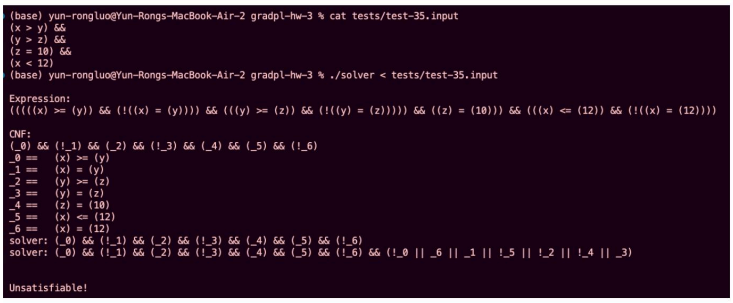

Figure 1: test-35

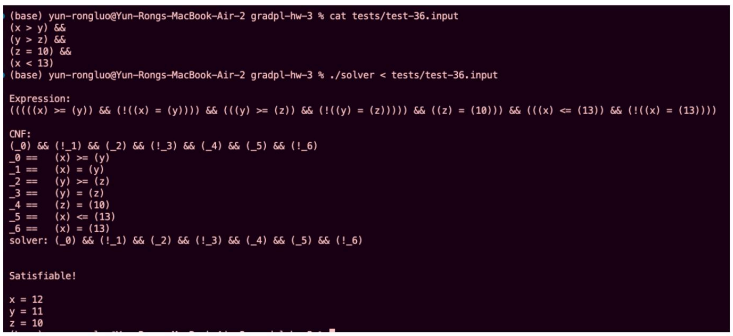

Figure 2: test-35

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 runtime bottleneck of DPLL(T) for these two cases lies in the theory solver. These two cases contain three variables that involve equalities and inequalities. DPLL(T) first performs propositional abstraction and encode 7 atoms of (in)equalities into 7 propositional variables as shown in the figures. A SAT solver then solves the propositional CNF abstraction for the given theory clauses. Since there are only 7 propositional variables, the simple DPLL SAT solver can handle it very efficiently. The arithmetic theory solver will check whether an abstract model returned by DPLL SAT solver is consistent to a concrete theory model. The arithmetic theory solver in this implementation simply enumerates all possible values from -127 to 128 for each variable, evaluates the values for the 7 atoms, and checks if the values are consistent with the abstract model. This step is the most time-consuming part. I would rewrite this part to improve performance. Whenever the value is decided for a variable, the theory solver should propagate the value in the theory clauses, and derive upper-bounds or lower-bounds for other undecided variables.