Questions assigned to the following page: 3 and 2

2 Exercise 3F-1. Regular Expressions,Large Step

The remaining three rules are concatenation(e_1e_2), or($e_1|e_2$), and kleene star(e^*).

Concatenation

$$\frac{\vdash e_1 \text{ matches } s_1 \text{ leaving } s_2 \quad \vdash e_2 \text{ matches } s_2 \text{ leaving } s'}{\vdash e_1e_2 \text{ matches } s_1 \text{ leaving } s'}$$

Or

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

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

Kleene Star

$$\frac{}{\vdash e_1^* \text{ matches } s \text{ leaving } s'}$$

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } s'}{\vdash e_1^* \text{ matches } s \text{ leaving } s'}$$

3 Exercise 3F-2. Regular Expressions and Sets

The two remaining rules are for concatenation and the Kleene star.

One attempt to write them might be the following:

Concatenation

$$\frac{\vdash e_1 \text{ matches } s_1 \text{ leaving } S \quad \vdash \exists s_2 \in S.e_2 \text{ matches } s_2 \text{ leaving } S'}{\vdash e_1e_2 \text{ matches } s_1 \text{ leaving } S \cap S'}$$

Kleene Star

$$\frac{}{\vdash e_1^* \text{ matches } s \text{ leaving } \{s\}}$$

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S}{\vdash e_1^* \text{ matches } s \text{ leaving } S}$$

However, upon application of these rules, it can be shown they are unsound, due to the branching behavior of the derivations.

For the concenation rule, any regex could be considered to be prefixed with null, so the output of any string would be every single character in the string. That would violate the usefulness or any kind of intuition behind regex.

For the Kleene star rule, it can be shown that "(ab)*, abab" can be derived to be both $\{abab\}$ or $\{ab, abab\}$, thus the rule is unsound.

Every intuitive rule to define Kleene star will require the ability to either stop or continue taking matches of the expression, thus it is not possible to define a complete and sound set of operational semantics for this language.

Questions assigned to the following page: 5 and 4

4 Exercise 3F-3. Equivalence

It has been formally shown that equivalence of regular expressions is decideable, as described by Andrea Asperti.

Intuitively, the respective DFAs of any two regexes can be merged into a DFA that accepts inputs that only match with one regex, and not with the other. It can be checked if this DFA accepts any inputs by performing reachability analysis. If it does not accept any inputs, then the regexes are equivalent.

5 Exercise 3F-4. SAT Solving

The last two tests consist of the following sets of inequalities:
Test 35: $(x > y) \land (y > z) \land (z = 10) \land (x < 12)$
Test 36: $(x > y) \land (y > z) \land (z = 10) \land (x < 13)$

Due to the and clauses, it is forced that every single arithmetic rule holds at once, there is no potential for other satisfying assignments with some of them being true and others false. Because there aren't multiple correct answers(in both SAT and the arithmetic theory), the search space is broader and our theory has a lower chance of finding the answer quickly.

To improve performance on this case, we need to improve the arithmetic theory, increased improvements on SAT will have minimal impact, as the complexity involves proving the multiple arithmetic clauses. The theory implementation provided exhaustively enumerates the possibilities of each variable, which makes it become exponentially slower with each increase in the number of variables. The algorithm could be improved by combining the transitive properties of each subsequent bound, and using that to further constrain the space. For example, the $(x > y) \land (y > z) \implies (x > y > z)$ could greatly shorten the joint search spaces for each the variables. As an algorithm, one could store an interval of each variable that is implied by the constraints, and collapse the intervals based upon the constraints that include a constant. Upon these much smaller intervals, a solution could be bruteforced in a faster time.