Questions assigned to the following page: 3 and 2

Exercise 3F-1. Regular Expression, Large-Step [10 points].
For concatenation e_1e_2, we have one rule that ensures e_1 matches s and e_2 matches the resulting s':

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

For $e_1|e_2$, we need two rules. If either of e_1 or e_2 matches s, we can say $e_1|e_2$ matches s:

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

For $e*$, we also need two rules. Notably, $e*$ matches even with 0 occurrences of e:

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

In the other case, we can take a recursive approach as $e*$ matches with as many occurrences of e as possible:

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

Exercise 3F-2. Regular Expression and Sets [5 points].
Rule for e_1e_2:

$$\frac{\vdash e_1 \text{ matches } s \text{ leaving } S \quad \vdash e_2 \text{ matches } s' \text{ leaving } S'}{\vdash e_1e_2 \text{ matches } s \text{ leaving } \{s''|s' \in S, s'' \in S'\}}$$

Rules for $e*$:

$$\frac{}{\vdash e* \text{ matches } s \text{ leaving } \{s\}} \qquad \frac{\vdash e \text{ matches } s \text{ leaving } S \quad \vdash e* \text{ matches } s' \text{ leaving } S'}{\vdash e* \text{ matches } s \text{ leaving } \{s''|s' \in S, s'' \in S'\}}$$

Questions assigned to the following page: 5 and 4

Exercise 3F-3. Equivalence [7 points].
We can show that $e_1 \sim e_2$ is undecidable by reducing from the halting problem. Given a Turing machine M and input x, we want to choose some regular expressions e_1 and e_2 such that if M halts on x, then $e_1 \sim e_2$ does not hold, and if M does not halt on x, then $e_1 \sim e_2$ holds.

Let us construct two regular expressions e_1, e_2 as so:

- Let e_1 be a regular expression that matches a string of the computational steps of M on x, which includes a halt condition if it exists.
- Let e_2 be a regular expression that also matches a string of the computational steps of M on x, but without any existing halt conditions.

With these definitions, we can see that if M halts on x, then e_1 will match a string with a halt, while e_2 will not. Therefore, $e_1 \sim e_2$ does not hold. Conversely, if M does not halt on x, then both e_1 and e_2 will match the same strings, both of which contain no halt condition. Therefore, $e_1 \sim e_2$ holds.

We have shown that if $e_1 \sim e_2$ were to be decidable, then the halting problem would as well. As the halting problem is undecidable, we have proven that $e_1 \sim e_2$ is undecidable as well.

Exercise 3F-4. SAT Solving [6 points].
The last two tests take a comparatively long time because the variables are dependent on each other, and as a result, each constraint results in several propagations. In the example of the test cases, once $z = 10$ is assigned, unit propagation also results in $y > 10$, $x > 10$, and so forth. Because of this, I think it would be best to revise the handle_unit_clauses function in dpll.ml, which handles unit propagation. Rather than the recursive approach given by handle_unit_clauses, an iterative approach that iterates over the clauses and handles unit propagation could be used instead. This could increase efficiency and prevent deep recursive stacks, particularly in situations that require extensive unit propagation such as the last two tests.