Questions assigned to the following page: 2, 3, 5, and 4

Exercise 3F-1. Regular Expression, Large-Step

$$
\begin{array}{rcl}
e & := & \texttt{"x"} \\
& | & \texttt{empty} \\
& | & e_1\ e_2 \\
& | & e_1 \mid e_2 \\
& | & e^* \\
& | & . \\
& | & [\texttt{"x"} - \texttt{"y"}] \\
& | & e+ \\
& | & e? \\
\\
s & ::= & \texttt{nil} \mid \texttt{"x"} :: s
\end{array}
$$

RE-Concatenation
$$\frac{\vdash s\ \texttt{matches}\ e_1\ \texttt{leaving}\ s_1 \qquad \vdash s_1\ \texttt{matches}\ e_2\ \texttt{leaving}\ s'}{\vdash s\ \texttt{matches}\ e_1\ e_2\ \texttt{leaving}\ s'}$$

RE-Or1
$$\frac{\vdash s\ \texttt{matches}\ e_1\ \texttt{leaving}\ s'}{\vdash s\ \texttt{matches}\ e_1 \mid e_2\ \texttt{leaving}\ s'}$$

RE-Or2
$$\frac{\vdash s\ \texttt{matches}\ e_2\ \texttt{leaving}\ s'}{\vdash s\ \texttt{matches}\ e_1 \mid e_2\ \texttt{leaving}\ s'}$$

RE-EKleene
$$\frac{}{\vdash s\ \texttt{matches}\ e^*\ \texttt{leaving}\ s}$$

RE-NEKleene
$$\frac{\vdash s\ \texttt{matches}\ e\ e^*\ \texttt{leaving}\ s'}{\vdash s\ \texttt{matches}\ e^*\ \texttt{leaving}\ s'}$$

Exercise 3F-2. Regular Expressions and Sets

I don't believe that it's possible to give a fully deterministic set of rules in the given system that captures the full set of possible matches. In `DRE-NEConcat` below we only match for a single s_1 in S therefore getting a subset of the potential matches; we cannot do it for all in S as it may be infinite and we may not have infinitely many hypothesis in our rule. `DRE-Kleene` on the other hand is unsound as the rule is recursive and there is no terminal case which would cause the derivations to have infinite length.

DRE-NEConcat
$$\frac{\vdash s\ \texttt{matches}\ e_1\ \texttt{leaving}\ S \qquad \exists s_1 \in S. \qquad \vdash s_1\ \texttt{matches}\ e_2\ \texttt{leaving}\ S'}{\vdash s\ \texttt{matches}\ e_1\ e_2\ \texttt{leaving}\ S'}$$

DRE-Kleene
$$\frac{\vdash s\ \texttt{matches}\ e\ e^*\ \texttt{leaving}\ S}{\vdash s\ \texttt{matches}\ e^*\ \texttt{leaving}\ \{s\} \cup S}$$

Exercise 3F-3. Equivalence

To show the decidability of the equivalence relation, we must note that it is an equivalent problem to show that their languages are equivalent—this is clear by considering that the equivalence relation is a match on all prefixes of $s \in S$. Regular expressions define regular languages and can therefore be defined by DFAs; DFAs can also be minimized and it can be shown that all equivalent DFAs have an isomorphic minimum representation. Therefore if we minimize both regular expression's DFAs we can check that they are isomorphic by going over all the states and transitions in both automata the desired equivalence relation can be shown.

Exercise 3F-4. SAT Solving

The last two examples are especially slow because of our naive integer constraint solving algorithm. Inputs 35 and 36 only go through DPLL one time with boolean assignments assigned for the arithmetic expressions so the constraint solver is taking the vast majority of the time by just trying the full search space of small ints restricting only to a given upper and lower bounds on variables. Worst case performance for DPLL/SAT solving it self is $O(2^n)$ but as noted before the boolean satisfiability is not our main issue in these examples. For inputs 35 and 36 we have 3 integer variables that have to fully enumerate their state spaces to prove satisfiability and unsatisfiability. For 36 specifically only the combination `z=10;y=11;x=12` satisfies the formula so arith.ml tests 256^3 possible test

Question assigned to the following page: 5

cases. The situation is similar for 35 except that there are *NO* satisfying solutions so the algorithm definitely tries all combinations. In general this means for an unsatisfiable expression the arithmetic constraint solving is 256^n so worst case is $O(256^n)$ where n is the number of arithmetic variables. The main defect in the arithmetic constraint solving is that it's just brute forcing the state space; this can be addressed in multiple ways: when a constraint is added into our current context we should bound the choice for all other variables to satisfy the given constraints—this is essentially making the arithmetic choose operator lazy the same way that DPLL(T) does. Another improvement similar to uniq propagation would be to take all unit arithmetic expressions in the conjunction and use them to bound variables, for input 35 that would immediately bind $z = 10; x < 12$ cutting down on the overall search space significantly. In addition I imagine there are several other heuristics that could be applied to improve speed such as specializing the order in which variables are chosen.

The last two examples are especially slow due to our naive integer constraint-solving algorithm. Inputs 35 and 36 only require one DPLL pass with boolean assignments for the arithmetic expressions. However, the constraint solver dominates the runtime by exhaustively searching the small integer space within the given variable bounds.

While the worst-case performance for DPLL/SAT solving is $O(2^n)$, the primary bottleneck here is the arithmetic solver. For inputs 35 and 36, three integer variables must fully enumerate their state spaces to determine satisfiability. Specifically, input 36 has only one satisfying assignment (`z=10;y=11;x=12`), requiring the solver to check 256^3 possibilities. Input 35 is unsatisfiable, forcing the solver to explore all combinations. In general, for n arithmetic variables, the worst-case complexity of the arithmetic solver is $O(256^n)$.

The core issue is that the solver relies on brute-force enumeration. Several optimizations could address this inefficiency. First, when a new constraint is added, we could dynamically restrict the domains of other variables—akin to how DPLL(T) handles theory propagation. Second, similar to unit propagation, we could use unit arithmetic constraints to immediately refine variable bounds. For instance, in input 35, this would immediately set $z = 10$ and $x < 12$, drastically reducing the search space. Other heuristics, such as optimizing the variable selection order, could further improve performance.