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

Exercise 3F-1.

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

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

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

$$\frac{\text{empty} | e e^* \text{ matches } s \text{ leaving } s'}{e^* \text{ matches } s \text{ leaving } s'}$$

Exercise 3F-2. This is not possible because for a concatenation $e_1 e_2$, the place where e_2 begins matching is determined by the number of characters e_1 consumes. e_1 may consume any number of characters so we must check if e_2 matches at every index, but this we cannot do because we need a fixed number of premises.

We could try

$$\frac{e_1 \text{ matches } s \text{ leaving } S \quad s' = min(S) \quad e_2 \text{ matches } s' \text{ leaving } S'}{e_1 e_2 \text{ matches } s \text{ leaving } S'}$$

This is deterministic and sound, but it is incomplete because "a"*"b" doesn't match "ab"

We could try

$$\frac{}{e_1 e_2 \text{ matches } s \text{ leaving } \{\}}$$

This is deterministic and complete, but it is unsound because "a""b" matches "".

Exercise 3F-3.
Convert both expressions into DFAs and use DFA minimization (Wikipedia article) to find their corresponding unique minimal automata. The two expressions are equivalent iff their corresponding minimal automata are structurally identical. This can be computed by brute force checking if a bijective mapping of states exists where all edges match.

Exercise 3F-4.
Both test case 35 and 36 contain three integer variables. The integer arithmetic solver being used is brute force (arguably an egregious defect), so it tries every possible assignment of variables. Since the range being considered is -127 to 128, 256^3 combinations of variables

Question assigned to the following page: 5

must be tested when the arithmetic solver is called with all three variables. The problem would be even worse if the arithmetic solver was called multiple times, but fortunately for both of these queries, DPLL(T) only calls the theory solver once. Case 36 is slightly faster than case 35 because case 36 short circuits when a satisfying assignment is found but case 35 is unsatisfiable so the full search space must be checked. This issue could be alleviated if with a smarter integer arithmetic solver.