1 3F-1 Bookkeeping

- 0 pts Correct

Exercise 3F-2. Regular Expression, Large-Step [10 points].

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

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

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

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

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

Exercise 3F-3. Regular Expression and Sets [5 points].

This can not be done.

- $\vdash e_1\ e_2$ matches s leaving S is not possible because the matching of e_2 is dependent on each of the suffixes after matching e_1. While the number of suffix is not fixed, so the matching of e_2 can not be expressed with fixed number of hypothesises.
- To express $\vdash e*$ matches s leaving S, it at least have to express the special case of $\vdash e\ e$ matches s leaving S. Applying same reason above here, we can prove that this special case is not expressible, which proves the general rule of $\vdash e*$ matches s leaving S is also not expressible.

Exercise 3F-4. Equivalence [7 points].

This is possible.

Since any regular expression can be converted to a DFA, and then reduced to a unique minimal-DFA, we just have to compare if the minimal-DFA is the same.

2 3F-2 Regular Expressions, Large Step

- 0 pts Correct

Peer Review ID: 68547174 — enter this when you fill out your peer evaluation via gradescope

Exercise 3F-2. Regular Expression, Large-Step [10 points].

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

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

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

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

$$\frac{\vdash e \textsf{ matches } s \textsf{ leaving } s' \qquad \vdash e* \textsf{ matches } s' \textsf{ leaving } s''}{\vdash e* \textsf{ matches } s \textsf{ leaving } s''}$$

Exercise 3F-3. Regular Expression and Sets [5 points].

This can not be done.

- $\vdash e_1\ e_2$ matches s leaving S is not possible because the matching of e_2 is dependent on each of the suffixes after matching e_1. While the number of suffix is not fixed, so the matching of e_2 can not be expressed with fixed number of hypothesises.
- To express $\vdash e*$ matches s leaving S, it at least have to express the special case of $\vdash e\ e$ matches s leaving S. Applying same reason above here, we can prove that this special case is not expressible, which proves the general rule of $\vdash e*$ matches s leaving S is also not expressible.

Exercise 3F-4. Equivalence [7 points].

This is possible.

Since any regular expression can be converted to a DFA, and then reduced to a unique minimal-DFA, we just have to compare if the minimal-DFA is the same.

3 3F-3 Regular Expressions and Sets

- 0 pts Correct

Exercise 3F-2. Regular Expression, Large-Step [10 points].

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

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

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

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

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

Exercise 3F-3. Regular Expression and Sets [5 points].

This can not be done.

- $\vdash e_1\ e_2$ matches s leaving S is not possible because the matching of e_2 is dependent on each of the suffixes after matching e_1. While the number of suffix is not fixed, so the matching of e_2 can not be expressed with fixed number of hypothesises.
- To express $\vdash e*$ matches s leaving S, it at least have to express the special case of $\vdash e\ e$ matches s leaving S. Applying same reason above here, we can prove that this special case is not expressible, which proves the general rule of $\vdash e*$ matches s leaving S is also not expressible.

Exercise 3F-4. Equivalence [7 points].

This is possible.

Since any regular expression can be converted to a DFA, and then reduced to a unique minimal-DFA, we just have to compare if the minimal-DFA is the same.

4 3F-4 Equivalence

- 0 pts Correct

Exercise 3F-4. SAT Solving [6 points].

- The main reason of the solver being slow is its bad integer arithmetic constraint solver part, namely the *arith.ml*, which will be the single module I will rewrite first. It is super slow because it brute-force all the possible value combination of all the variables. It is obvious that we do not have to iterate all the possible values of x to figure out how to satisfy the literal $x = 1$. There are smarter algorithms to implement *arith.ml*. For example if we are solving only linear integer arithmetic, I will follow this paper: Jovanović and Moura 2013; for more complicated non-linear integer arithmetic, I will probably start from here: Cimatti et al. 2018.
- Another reason is that this implementation is a lazy-version of the DPLL(T). The main difference is how the SAT solver interacts with the theory solver(or the arithmetic constraint solver in our case).
  - For our lazy version, the SAT solver spits out the whole possible assignment, then the theory solver check if that is consistent, if not add the negation of the assignment back to the initial clause and start over again. This is slow because all the theory information is not used to guide the search. For example for clause $x = 5 \ \wedge \ x \neq 5 \ \wedge (\textit{something complicated}) \ \wedge ...$, the SAT solver will keep generating different assignments for the trialing complicated clauses but always assign both $x = 5$ and $x \neq 5$ to be true, while the theory solver will keep pushing back the negation of all these assignments, until they tried all the combinations.
  - The DPLL(T), on the contrary, have several optimizations for that.
    - Check the consistency of the partial assignments while being built. This means instead of having SAT propose a full assignment, SAT solver will ask the theory solver to check each step it makes. This allows early termination of impossible search directions.
    - When encounter inconsistency, instead of adding the negation of the whole assignment, we ask the theory solver to narrow down a inconsistent subset and add the negation of that. This will enable a faster elimination of the impossible assignments.
    - After adding the negation clause, instead of starting over, we only backtrack to the point where is still consistent with the added clause. This will potentially save some of the repeated initial assignments like the unit clauses.

  All of these optimizations are different ways to use the theory information to direct the SAT search, which makes DPLL(T) much faster than our lazy implementation. However, implementing that is a lot of extra work. We have to rewrite our SAT solver and theory solver to be incremental and backtrack-able and also rewrite the main file fit to in all the incremental checking and backtracking control flows.

References

[Cim+18] Alessandro Cimatti et al. “Experimenting on Solving Nonlinear Integer Arithmetic with Incremental Linearization”. In: June 2018, pp. 383–398. ISBN: 978-3-319-94143-1. DOI: 10.1007/978-3-319-94144-8_23.

[JM13] Dejan Jovanović and Leonardo de Moura. “Cutting to the Chase: Solving Linear Integer Arithmetic (to appear)”. In: *Journal Automated Reasoning* (2013).

5 3F-5 SAT Solving

- 0 pts Correct