CS 3100 – Models of Computation – Fall 2011

FIRST MIDTERM – CLOSED BOOK – 100 points

I've standardized on "@" for representing Epsilons in all my figures as well as my code (liked equally by `dot` and our Python programs).

1. **(20 pts)** Convert this NFA N into an equivalent DFA D

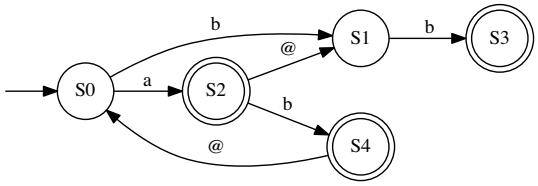

Figure 1: NFA N

Answer: See Figure 2.

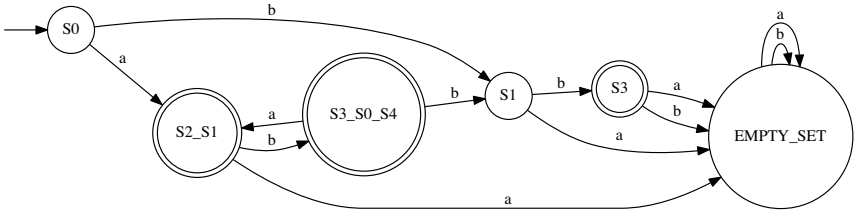

Figure 2: Equivalent DFA N obtained through function `nfa2dfa`

2. **(25 pts)** Using the GNFA method, convert N into an equivalent RE R. **Eliminate states in the order S3, S1, S4, S2, S0.**

   Answer: See Figures 4 through 9. The RE can be read off Figure 9. To further validate that this RE is correct, we provide Figures 10 which turns this RE into an NFA using function `re2nfa` in `reparse.py`, and then 11 using `re2dfa`, and finally 12 using `re2mindfa`.

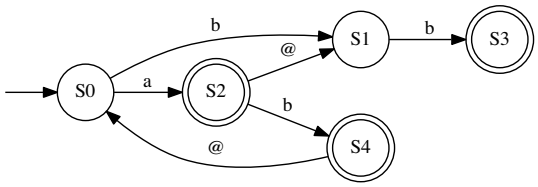

Figure 3: NFA N repeated for your convenience

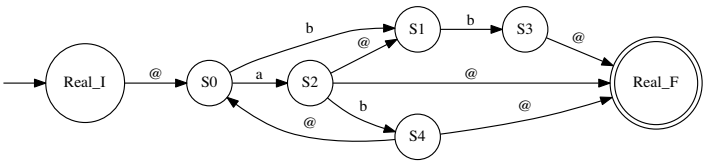

Figure 4: Initial GNFA obtained through `mk_gnfa_N`

the final RE is `(ab)* (a+ab+bb)`

3. **(25 pts)** Minimize this DFA, showing the steps.

   Show your minimization table with its "X_n" entries for $n = 0, 1, 2, \ldots$ here. Also draw the minimized DFA on this page.

   Answer: The minimization is straightforward (ask us if you are unsure). The table of minimization is as shown in Figure 14.

   See Figure 15 and 16 for the minimized DFAs (the latter showing the merged equivalence classes).

4. **(3 pts)** What is `list(map(lambda y:2*y, map(lambda x:x+1, [30, 21, 31])))`

   Answer: `[62, 44, 64]`

5. **(3 pts)** What is the concatenation of $\{a, ab, \varepsilon\}$ and $\{a, b, \varepsilon\}$?

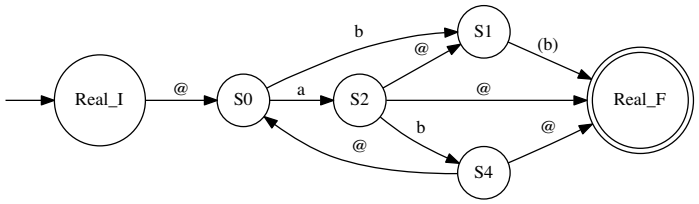

Figure 5: Result of eliminating S3 from Figure 4, obtained using `del_state_from_gnfa_N`

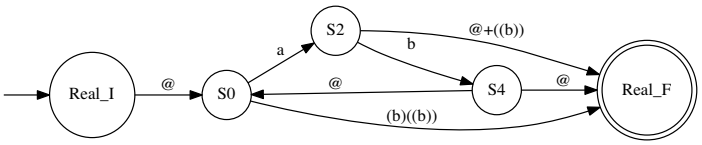

Figure 6: Result of eliminating S1 from Figure 5, obtained using `del_state_from_gnfa_N`

Answer: $a, ab, aba, abb, b, \varepsilon$

6. **(3 pts)** Draw an NFA corresponding to the RE `(a+ab)* (b+ab)*` (concatenation of `(a+ab)*` and `(b+ab)*`) using the standard approach. *Just show the final NFA, but don't simpilify anything* (to facilitate grading).

   Answer: The final answer was obtained using `re2nfa('(a+ab)* (b+ab)*')`, as in Figure 17.

7. **(3 pts for EACH string)** Write down two strings *not* in the language of `(a+ab)* (b+ab)*`, providing the reasons.

   Answer: We must begin the construction involving the first RE. But the first RE itself can't end in an `a` after a `b`, so we must employ the second RE. But the second RE can't end in an `a`.

8. **(2 pts)** Draw a DFA D_3 for the language of strings over $\{0, 1\}$ that, when interpreted as binary numbers, are divisible by 3 without remainder (the "mod-3 language").

   Answer: Figure 15 gives this!

9. **(2 pts)** Now draw the Kleene-star of D_3 as an NFA using the standard construction, drawing it as N_3^*.

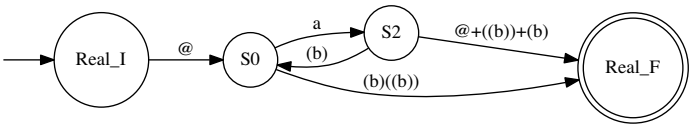

Figure 7: Result of eliminating S4 from Figure 6, obtained using `del_state_from_gnfa_N`

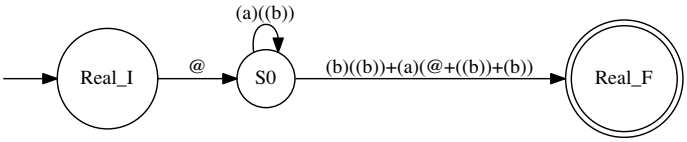

Figure 8: Result of eliminating S2 from Figure 7, obtained using `del_state_from_gnfa_N`

Answer: We essentially obtain back Figure 15!

10. **(3 pts)** Now convert N_3^* by inspection to a DFA D_3^*. What is the relationship between D_3^* and D_3, and why is there such a relationship (answer with respect to *the languages* of these machines)?

    Answer: We essentially obtain back Figure 15! Concatenating strings divisible by 3 is tantamount to weighing these strings by powers of 2 and adding. This does not change divisibility by 3.

11. **(3 pts)** Draw the NFA N for a language L for which its DFA D is exponential (with respect to the NFA-size); in addition, for the reverse of this language L, there must exist a DFA that is linear in size.

    Answer: This is the language where the N-th from last is a 1.

12. **(2 pts)** Explain in one sentence why the DFA D of Question 11 is exponentially sized.

    Answer: The DFA must keep track of the last N bits in all combinations, to accurately know, at each point, what the Nth-last bit was.

13. **(3 pts)** Why is the language of regular expressions over any alphabet (when viewed as strings) not regular?

    Answer: It is context-free due to needing paranthesis matching (`(..)`), which requires counting, which can't be done using an FSA.

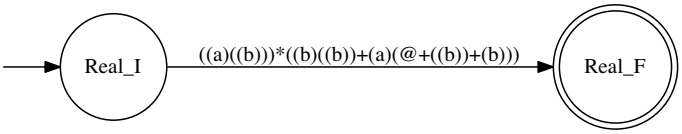

Figure 9: Result of eliminating S0 from Figure 8, obtained using `del_state_from_gnfa_N`

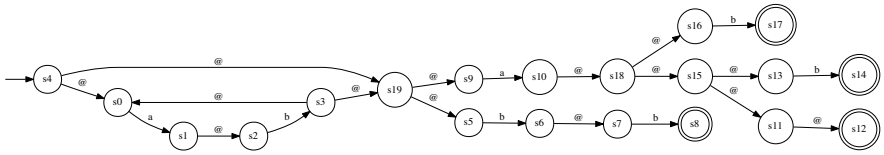

Figure 10: Apply `re2nfa` to the RE of Figure 9

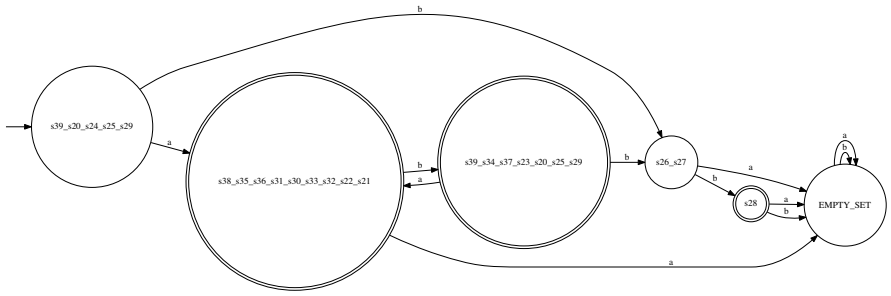

Figure 11: Apply `re2dfa` to the RE of Figure 9

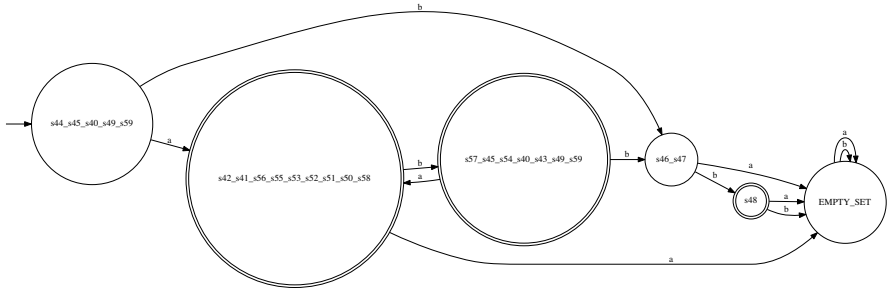

Figure 12: Apply `re2mindfa` to the RE of Figure 9

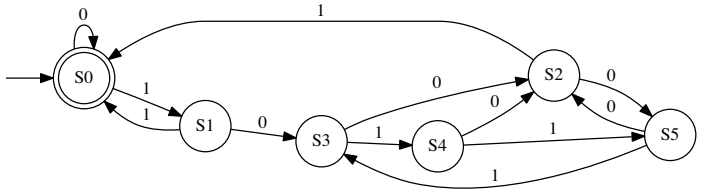

Figure 13: DFA to be minimized

```
 |
1|X0
 |
2|X0  .
 |
3|X0  X2  X3
 |
4|X0  X2  X3   .
 |
5|X0  X2  X3   .   .
-|-----------------------------------------
 | 0   1   2   3   4
```

Figure 14: DFA minimization table for the DFA in Figure 13

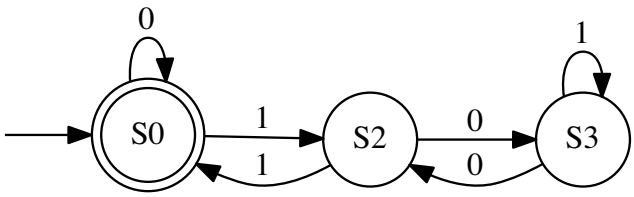

Figure 15: Minimized DFA with representative states obtained through `minDFA(mk_q3_dfa(), 'brief')`

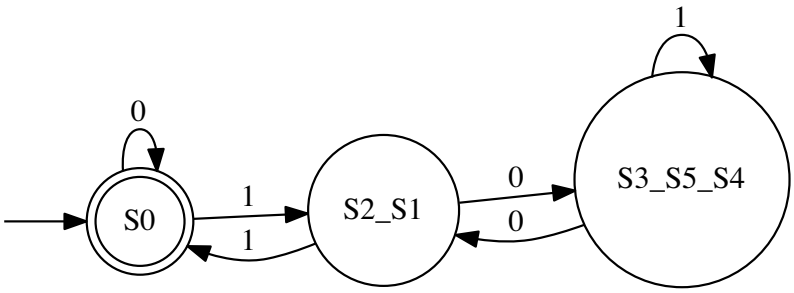

Figure 16: Minimized DFA with representative states obtained through `minDFA(mk_q3_dfa(), 'verbose')`

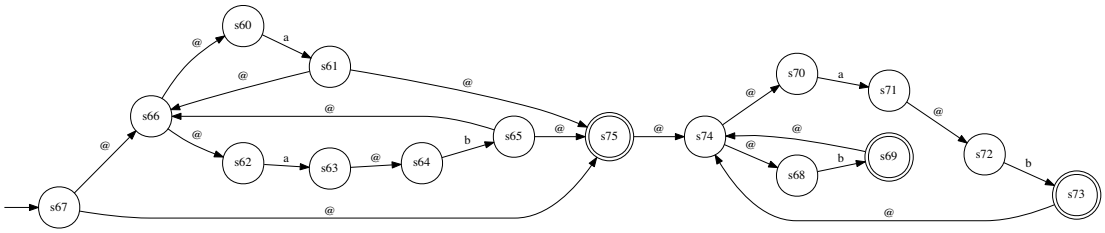

Figure 17: NFA obtained through `re2nfa('(a+ab)* (b+ab)*')`