CSE 311: Foundations of Computing I Winter 2025

Problem Set 8

Due: Thursday, June 5th by 11:00pm

Instructions

Submit your solutions in Gradescope. Your Gradescope submission should follow these rules:

- Each numbered task should be solved on its own page (or pages). Do not write your name on the individual pages. (Gradescope will handle that.)
- When you upload your pages, make sure each one is **properly rotated**. If not, you can use the Gradescope controls to turn them to the proper orientation.
- Follow the Gradescope prompt to **link tasks to pages**. You do not need to link tasks that you did not include, e.g., Tasks 1a-c & 2 (grin submission only) and Task 7 (extra credit).
- You are not required to typeset your solution, but your submission must be **legible**. It is your responsibility to make sure solutions are readable — we will *not* grade unreadable write-ups.

Task 1 – McDFAnalds [20 pts]

The TA's are quickly growing tired of fast food – mainly, because they keep getting into these absurd situations involving change. Luckily for them, this week, things are different... yes, they've actually managed to plan our order this time! However, they're experiencing one slight hiccup: Their language has become... irregular... and they can no longer figure out how to order! They have encoded their language debacle into several DFA-type problems, and they need your help to order their meals. Moreover, the TA's feel bad – every week, they promise fast food and never deliver. As such, they promise that this week the rewards will be immense; they just can't say what the rewards are for reasons out of their control (or something). Either way, please help! It's almost their turn to order...
For each of the following, create a *DFA* that recognizes exactly the language given.

a) Binary strings that start with 110 and have an odd number of 0's

b) Binary strings where every 0 is followed by an odd number of 1s.

c) Binary strings with exactly three 1's that also end with exactly two 0's.

d) Let $\Sigma = \{\wedge, \vee, \rightarrow, \neg\} \cup \{A, B\}$. If these symbols look familiar, they are! These are the symbols of **propositional logic** (minus parenthesis). In this problem, we will create a DFA for a subset of the **Well-Formed Formulas** of propositional logic! Particularly, we are going to be focusing on WFFs that do not have any sort of parenthesis (which is a pretty substantial set of WFFs!), which are all still valid formulas due to precedence rules. The reason we have left out parenthesis is that these require a *CFG*, due to the fact that we'd have to keep track of how many parenthesis were placed down (You will prove that parenthesis balancing is an *irregular* language in part 6a).

In CSE311, we have been primarily operating in the **object language** of propositional logic. However, if we want to prove statements about propositional logic itself, we can't use the same symbols! That is where the **metalanguage** comes in, which has all sorts of weird symbols, like $\implies, \models, \vdash$, and $\doteq$. For our sake, we will avoid metalanguage, but we can still do plenty of metalogic without metalanguage! After completing this problem, we will have quick and efficient way of checking if a propositional logic formula is valid regardless of the actual truth of any of the variables.

To create these formulas, we adhere to the following rules:

(a) A and B are our two propositional variables - we don't need any other ones for this problem.

(b) $\wedge, \vee, \rightarrow$ must be preceded by exactly one propositional variable, and followed by either a propositional variable or $\neg$. $A \wedge B \wedge \neg A$ is valid, but $A\wedge$ on its own is not. $A \rightarrow B \vee A$ is valid, but $A \rightarrow B\vee$ is not.

(c) $\neg$ must be followed by a propositional variable, or another $\neg$, $\neg\neg A \rightarrow B$ is valid, $\neg\neg \rightarrow B$ is not.

In other words, some examples of proper WFFs include: "$A \wedge A \wedge \neg B$", "$A \rightarrow \neg A \rightarrow B$", and "$A \vee B \rightarrow A \wedge B$".

(The following fact is not testable material)
Have you ever wondered why $A \implies B$ is not the same thing as $A \rightarrow B$? Well, you're in luck – today you'll finally find out! $A \implies B$ is a metalogical (semantic) statement, while $A \rightarrow B$ is an object-language (syntactic) statement, which is why we're such sticklers about the two seemingly

identical arrows not being the same. Thanks to Gödel's **Completeness Theorem**, if $A \implies B$ in the metalayer (written $A \models B$), then, in any sound and complete system of logic, $A \to B$ is provable (written $A \vdash B$). This is precisely how the **direct proof rule** works in practice!

Congrats! Not only are you an incredible logician, but you are now a certified metalogician!

Submit and check your answers to parts **(a-c)** here:

http://grin.cs.washington.edu

Think carefully about your answer to make sure it is correct before submitting. You have only **5 chances** to submit a correct answer.

Task 2 – Design Intervention [15 pts]

For each of the following, create an *NFA* that recognizes exactly the language described.

a) Binary strings with at least two 1s ending with a 0 **or** at least two 0s ending with a 1.

b) Binary strings that contain 11 but do not contain the substring 00.

c) Binary strings that start with 10, end in 01, **and** contain at most three 0s in total. Notice that 101 is in the language.

Submit and check your answers to this question here:

http://grin.cs.washington.edu

Think carefully about your answer to make sure it is correct before submitting. You have only **5 chances** to submit a correct answer.

Task 3 – The Great Expression [10 pts]

Use the algorithm from lecture to convert each of the following regular expressions into NFAs that accept the same language. You should **precisely** follow the construction from lecture.

a) $10(01 \cup 1)^*11 \cup 0110$

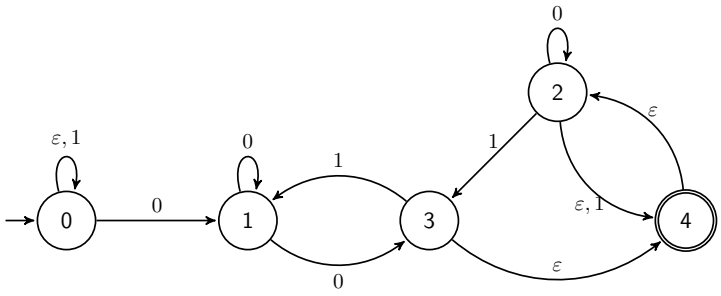
(Note: We translated this into a CFG in HW7 Task 6(a). Here, we translate it into an NFA.)

b) $(1(01 \cup 10)^*)^*$

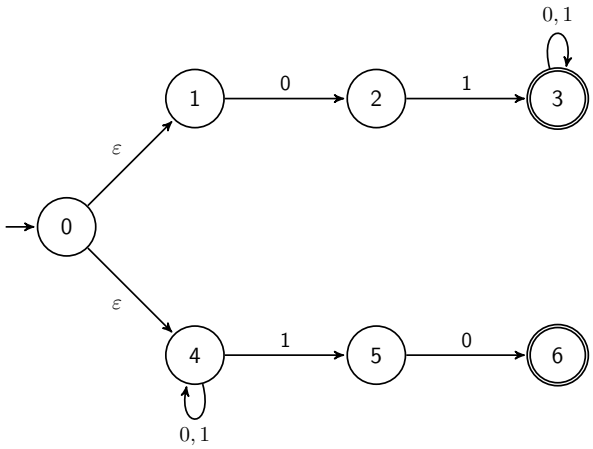
Task 4 – Final Study Mission [10 pts]

Use the algorithm from lecture to convert each of the following NFAs to DFAs. Label each DFA state with the set of NFA states it represents in the powerset construction.

a) The NFA below, which accepts strings starting with any number of 1's, immediately followed by "00", and then any binary string:



b) The NFA below, which accepts strings starting with "01" or ending in "10":



Submit and check your answers to this question here:

http://grin.cs.washington.edu

Think carefully about your answer to make sure it is correct before submitting. You have only **5 chances** to submit a correct answer.

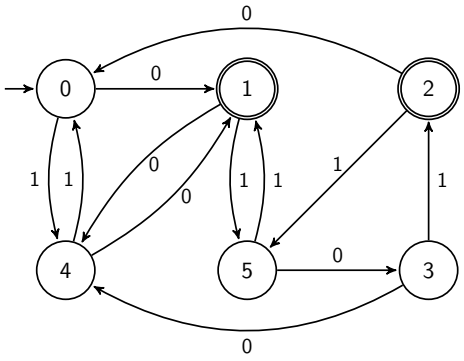
Note: You must also include a screenshot of each DFA in your submission to Gradescope so that we can verify that you properly named each of the states as a subset of the NFA states.

Task 5 – A Whole New Small Game [16 pts]

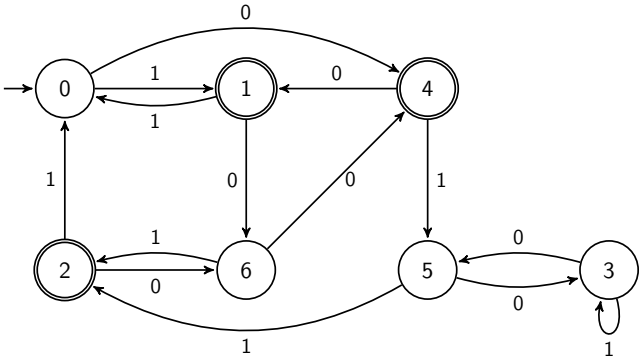
Use the algorithm from lecture to minimize the each of the following DFAs.

For each step of the algorithm, write down the groups of states, which group was split in that step and the reason for splitting that group. At the end, write down the minimized DFA, with each state named by the set of states of the original machine that it represents (e.g., "$\{B, C\}$").

a) The following machine:



b) The following machine:



Task 6 – Puedo ir(regular) al baño [20 pts]

Use the method described in lecture to prove that each of the following languages is **not regular**. You may need to get creative when constructing your prefix set and suffix!

a) The set of all properly balanced parentheses. For example, the following are balanced parentheses: "()()()", "((()))", "(()(()))". The following are unbalanced parentheses: "()()(", "((()))", "(((()((". Here is a CFG which reconizes balanced parentheses: $\mathbf{S} \to (S)|SS|\varepsilon$.

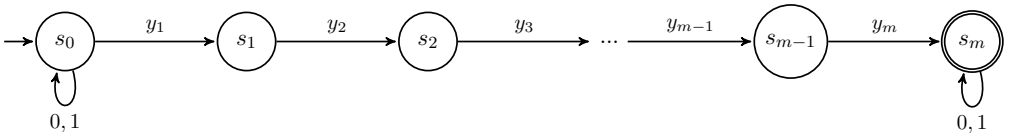
Side notes: Practically speaking, this is a basic language that any syntax checker (e.g. for Java, Python, even arithmetic expressions with parentheses, etc.) should be able to "recognize" to determine if your code is syntactically invalid! Whatever model of computation which does this must recognize more than regular languages. Also, recall from Task 1d that you built a DFA for propositional logic statements, but *without parentheses*, relying on precedence rules for interpretation of statements. When you solve this problem, you will have proven that it was impossible to replicate Task 1d for the set of all parenthesized propositional logic statements! Pretty neat — CS theory rocks!

b) All binary strings in the set $\{0^m1^n : m, n \in \mathbb{N} \text{ and } m \mid n\}$ (that is, all binary strings in which the number of 0's divides the number of 1's). *Hint: Think about the prime numbers, of which there are infinitely many.*

c) All binary strings in the set $\{0^m1^n : m, n \in \mathbb{N} \text{ and } m \leqslant n^2\}$ (that is, all binary strings in which the number of 0's is less than or equal to the square of the number of 1's).

Task 7 – Extra Credit: Strings to Mind [0 pts]

Suppose we want to determine whether a string x of length n contains a string $y = y_1 y_2 \dots y_m$ with $m \ll n$. To do so, we construct the following NFA:



(where the ... includes states $s_3, \dots, s_{m-2}$). We can see that this NFA matches x iff x contains the string y.

We could check whether this NFA matches x using the parallel exploration approach, but doing so would take $O(mn)$ time, no better than the obvious brute-force approach for checking if x contains y. Alternatively, we can convert the NFA to a DFA and then run the DFA on the string x. *A priori*, the number of states in the resulting DFA could be as large as 2^m, giving an $\Omega(2^m + n)$ time algorithm, which is unacceptably slow. However, below, you will show that this approach can be made to run in $O(m^2 + n)$ time.

(a) Consider any subset of states, S, found while converting the NFA above into a DFA. Prove that, for each $1 \leqslant j < m$, knowing $s_j \in S$ *functionally determines* whether $s_i \in S$ or not for each $1 \leqslant i < j$.

(b) Explain why this means that the number of subsets produced in the construction is at most $2m$.

(c) Explain why the subset construction thus runs in only $O(m^2)$ time (assuming the alphabet size is $O(1)$).

(d) How many states would this reduce to if we then applied the state minimization algorithm?

(e) Explain why part (c) leads to a bound of $O(m^2 + n)$ for the full algorithm (without state minimization).

(f) Briefly explain how this approach can be modified to count (or, better yet, find) *all* the substrings matching y in the string x with the same overall time bound.

Note that any string matching algorithm takes $\Omega(m + n) = \Omega(n)$ time in the worst case since it must read the entire input. Thus, the above algorithm is optimal whenever $m^2 = O(n)$, or equivalently, $m = O(\sqrt{n})$, which is the case for normal inputs circumstances.