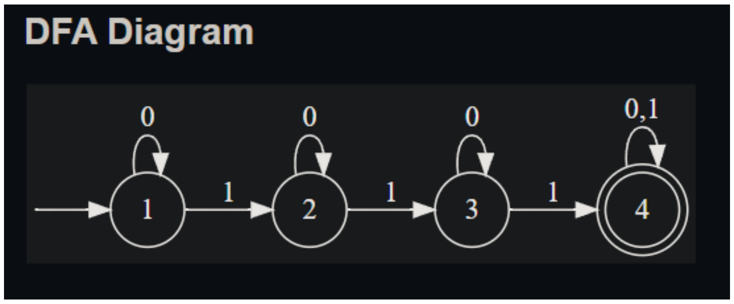
CPSC-439-01 Project #1 Write-up
Group 2 | Spring '23

Group Members:
- Yao Lin
- Justin Stitt
- Aaron Lieberman
- Long Vu
- Santiago Zavala
- Anthony Lam

Disclaimer: *Source files (markdown or Python) can be found in our Group's submission tar file.*
Note: *We named our files "TaskX.xx" instead of "StepX.xx".*

Step 1:
When asked to choose a language presented in Exercise 2.13 of "Introduction to Theory of Computation" we opted for the first option: {w: w contains at least three 1s}. Frankly, we believed this to be the easiest.

We used GraphViz to render the DFA. Since the grader may not have GraphViz installed I've included a screen capture below.



Step 2:
Next, we had to implement a DFA for the language. We chose **Python** due to our group's proficiencies with the language. The implementation, after having created our DFA, was trivial. Source code is provided in our submission's tar file.

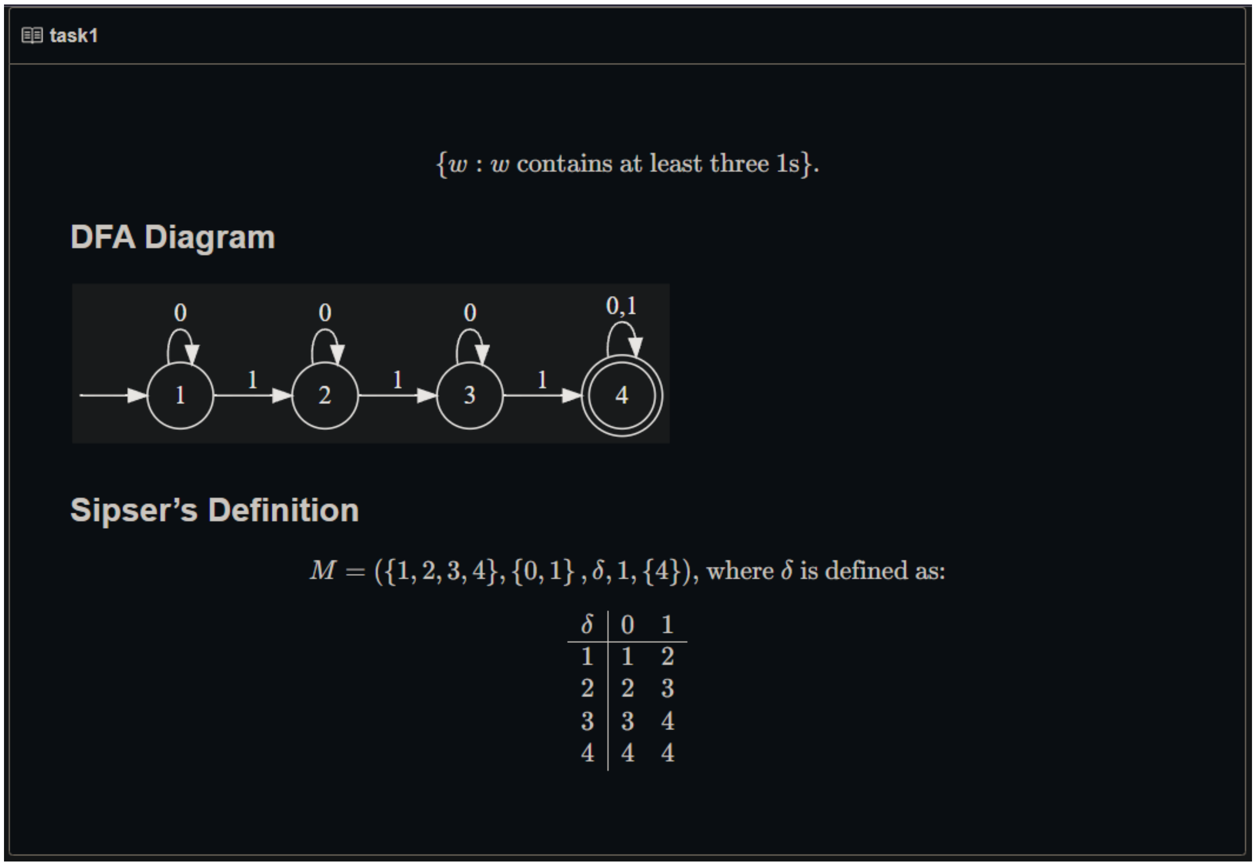
Step 3:
Next up, we had to design a Turning Machine implementation to recognize our previously chosen language. Admittedly, this part was also trivial after having created both our 1) DFA and 2) Python script. The final design is short and is adequately described in our submission tar file under the file name "task3.md"

Step 4:
Now, we had several ideas for writing the implementation for a Turing Machine to recognize our chosen language. However, we finally opted to go for a Python-based implementation – yet again. Our final working implementation can be found under "task4.py".

Step 5:
To define a language similar to our previously chosen language that is not regular, we just Googled "simple irregular languages". We stumbled upon a palindrome language which sounded quite fun and non-trivial. It was, however, trivial! Utilizing the Pumping Lemma technique we were able to demonstrate that the language is irregular. This is found under "task5.md" and requires a decent LaTeX renderer (GitHub's is fine).

Step 6:
Lastly, we had to implement a Turing Machine to recognize our irregular palindrome language. This can be found under the directory `./task6`. The design process was simple due to our strong understanding of both Turning Machines as well as palindromes. We wrote unit tests which can be ran with `$ python3 ./task6/tests.py`. You can, optionally, choose to run a case-by-case runthrough with `$ python3 ./task6/task6.py -i "aba"` which will return it's judgement on your input string "aba".

Attached below are some screen captures of our Markdown/LaTeX renderings – just in case the grader lacks a Markdown/LaTeX renderer. The task number is labelled in the top left of each image.



task1

$$\{w : w \text{ contains at least three 1s}\}.$$

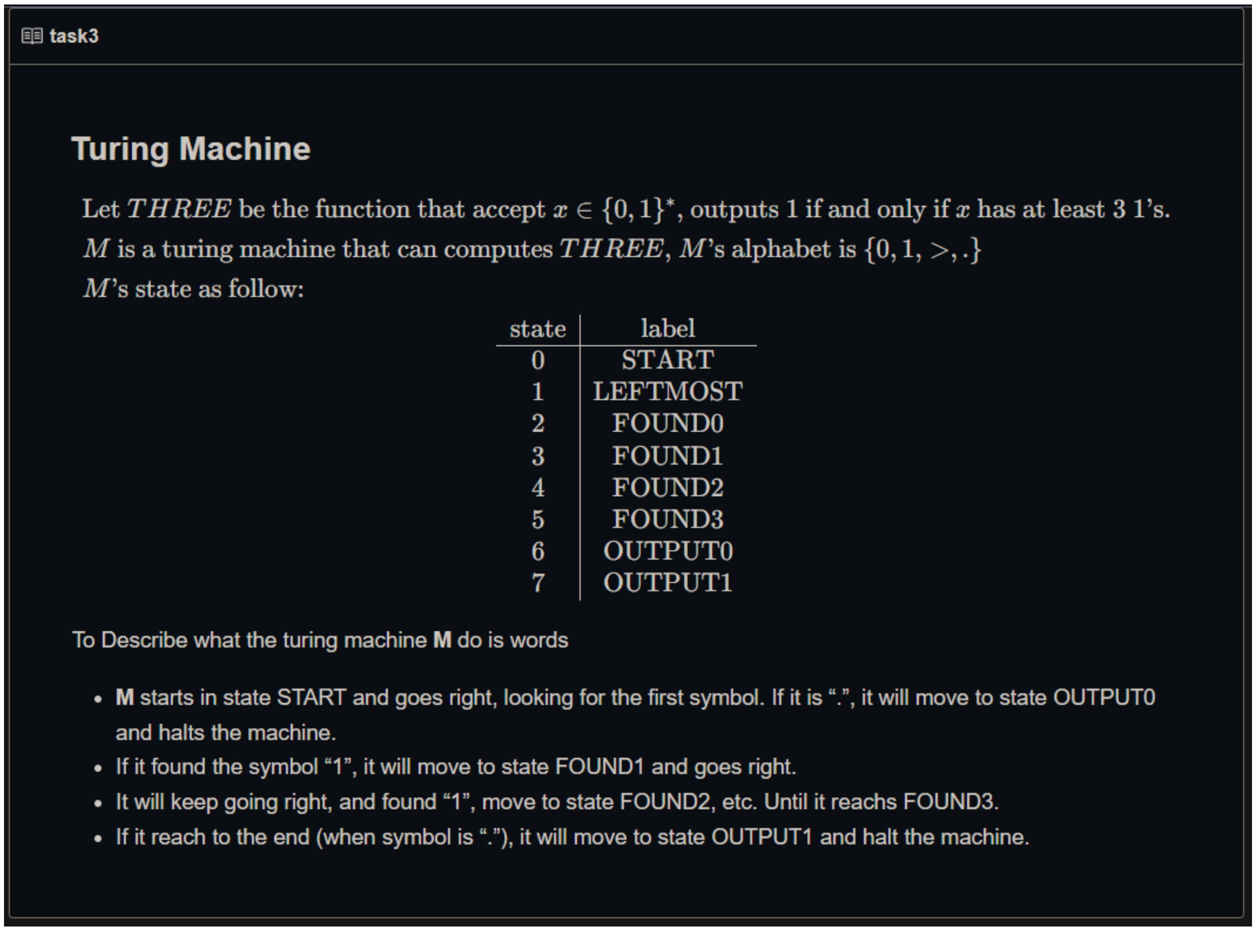
DFA Diagram

$$\begin{tikzcd}
\arrow[r] & 1 \arrow[loop above, "0"] \arrow[r, "1"] & 2 \arrow[loop above, "0"] \arrow[r, "1"] & 3 \arrow[loop above, "0"] \arrow[r, "1"] & \boxed{4} \arrow[loop above, "0,1"]
\end{tikzcd}$$

Sipser's Definition

$$M = (\{1,2,3,4\}, \{0,1\}, \delta, 1, \{4\}), \text{ where } \delta \text{ is defined as:}$$

δ	0	1
1	1	2
2	2	3
3	3	4
4	4	4


task3

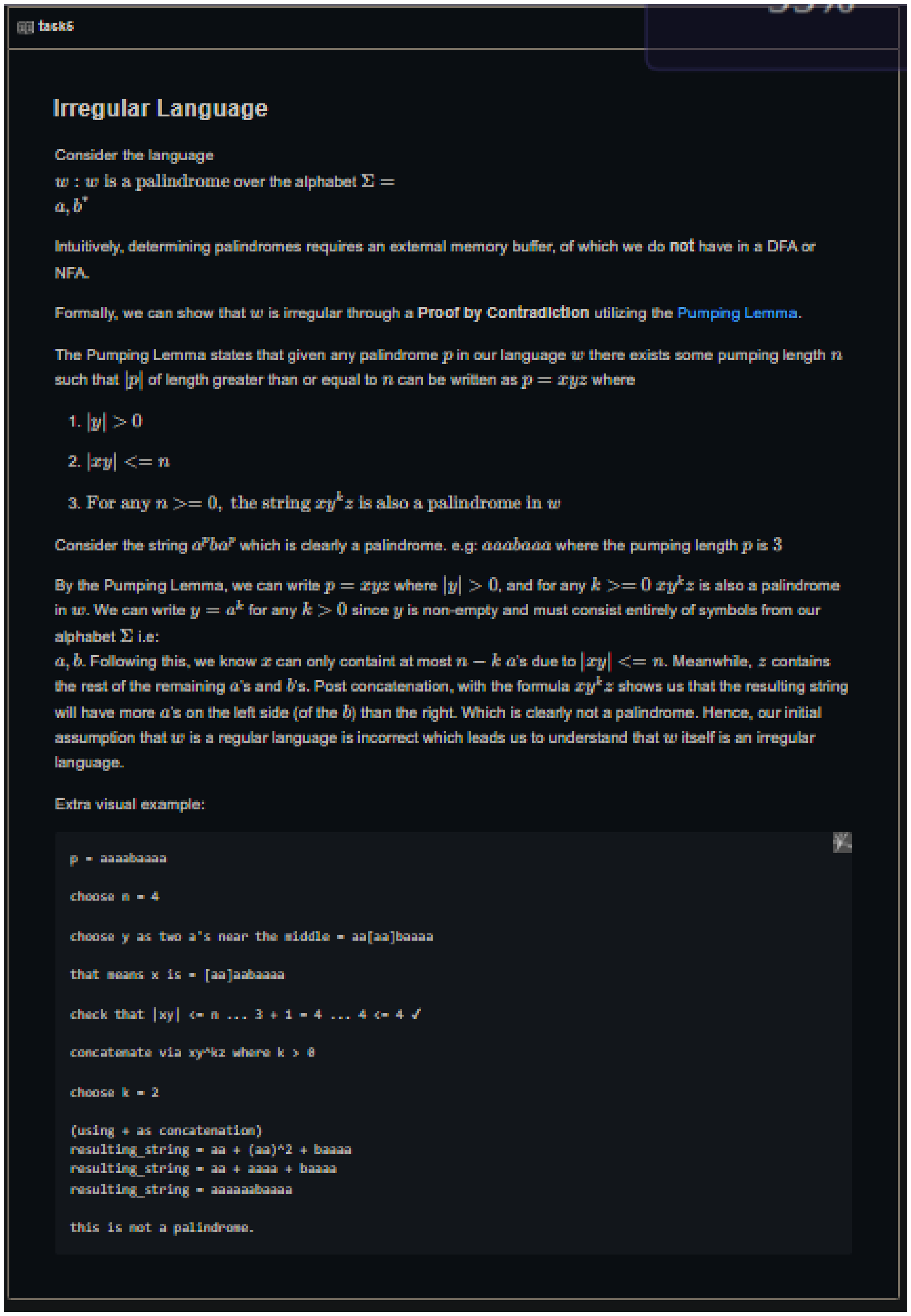
Turing Machine

Let $THREE$ be the function that accept $x \in \{0,1\}^*$, outputs 1 if and only if x has at least 3 1's.
M is a turing machine that can computes $THREE$, M's alphabet is $\{0, 1, >, .\}$
M's state as follow:

state	label
0	START
1	LEFTMOST
2	FOUND0
3	FOUND1
4	FOUND2
5	FOUND3
6	OUTPUT0
7	OUTPUT1

To Describe what the turing machine **M** do is words

- **M** starts in state START and goes right, looking for the first symbol. If it is ".", it will move to state OUTPUT0 and halts the machine.
- If it found the symbol "1", it will move to state FOUND1 and goes right.
- It will keep going right, and found "1", move to state FOUND2, etc. Until it reachs FOUND3.
- If it reach to the end (when symbol is "."), it will move to state OUTPUT1 and halt the machine.


task5

Irregular Language

Consider the language
$w : w \text{ is a palindrome}$ over the alphabet $\Sigma =$
a, b^*

Intuitively, determining palindromes requires an external memory buffer, of which we do **not** have in a DFA or NFA.

Formally, we can show that w is irregular through a **Proof by Contradiction** utilizing the Pumping Lemma.

The Pumping Lemma states that given any palindrome p in our language w there exists some pumping length n such that $|p|$ of length greater than or equal to n can be written as $p = xyz$ where

1. $|y| > 0$
2. $|xy| <= n$
3. $\text{For any } n >= 0, \text{ the string } xy^kz \text{ is also a palindrome in } w$

Consider the string a^pba^p which is clearly a palindrome. e.g: $aaabaaa$ where the pumping length p is 3

By the Pumping Lemma, we can write $p = xyz$ where $|y| > 0$, and for any $k >= 0$ xy^kz is also a palindrome in w. We can write $y = a^k$ for any $k > 0$ since y is non-empty and must consist entirely of symbols from our alphabet Σ i.e:
a, b. Following this, we know x can only containt at most $n - k$ a's due to $|xy| <= n$. Meanwhile, z contains the rest of the remaining a's and b's. Post concatenation, with the formula xy^kz shows us that the resulting string will have more a's on the left side (of the b) than the right. Which is clearly not a palindrome. Hence, our initial assumption that w is a regular language is incorrect which leads us to understand that w itself is an irregular language.

Extra visual example:

```
p = aaaabaaaa

choose n = 4

choose y as two a's near the middle = aa[aa]baaaa

that means x is = [aa]aabaaaa

check that |xy| <= n ... 3 + 1 = 4 ... 4 <= 4 ✓

concatenate via xy^kz where k > 0

choose k = 2

(using + as concatenation)
resulting_string = aa + (aa)^2 + baaaa
resulting_string = aa + aaaa + baaaa
resulting_string = aaaaaabaaaa

this is not a palindrome.
```