CS 3823 - Theory of Computation: Homework Assignment 4

Fall 2025 **Due:** Friday, November 14, 2025

Related Reading. Sections 2.2, 2.3. Chapter 3.
Instructions. Near the top of the first page of your solutions please list clearly **all** the members of the group (please see the syllabus for the collaboration policy) who have created the solutions that you are submitting. Listing the names of the people in the group implies their full name and their 4x4 IDs. Alternatively, you can use the space below and provide the relevant information in case you submit the solutions using this document.

	Lastname, Firstname	4x4 ID (e.g., dioc0000)
1	**Frison, Colby**	**fris0010**
2	**Ward, Levin**	**ward0209**
3	**Wage, Edward**	**wage0008**
4	**Mosisa, Joy**	**mosi0010**
5	**Wheeler, Ben**	**whee0113**

Grade

Exercise	Pages	Your Score	Max
1	2		8
2	3		8
3	4		8
4	5		8
5	6-7		8
Total	2-8		40

Additional Help and Resources. Did you use help and/or resources other than the textbook? Please indicate below.

1 Context-Free Grammar to Pushdown Automaton [8 points]

Consider the context-free grammar $G = (V, \Sigma, R, S)$ with $V = \{S, T\}, \Sigma = \{a, b, c\}$, and productions

$$\begin{array}{rcl} S & \to & aSc \mid T \\ T & \to & bTc \mid \varepsilon \end{array}$$

(i) **[2 points]** Describe in words (and set notation) the language $L(G)$.

(ii) **[6 points]** Construct a **pushdown automaton (PDA)** M that accepts $L(G)$. Please use the construction that we have seen in class (also discussed in the book). As usual, the PDA should start with an empty stack and accept strings in the language with an empty stack.

(i) Language Description

The grammar generates strings that have some number of leading a's, followed by some number of b's, and then as many trailing c's as the combined count of a's and b's. In formal notation:

$$L(G) = \{a^i b^j c^{i+j} \mid i, j \geqslant 0\}$$

We can understand how this works by looking at the grammar rules:

- The rule $S \to aSc$ adds one a to the front and one c to the back each time it's used.
- The rule $S \to T$ passes control to T when we're done adding a's and c's.
- The nonterminal T generates $b^j c^j$ by repeatedly using $T \to bTc$ and eventually $T \to \varepsilon$. This also allows for the empty string when both $i = 0$ and $j = 0$.

(ii) PDA Diagram

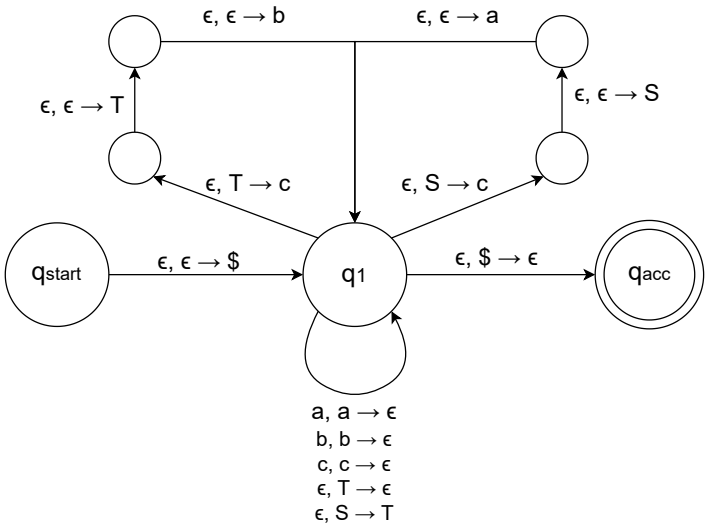

Figure 1: PDA Diagram

2 Pumping Lemma for Context-Free Languages [8 points]

Let $\Sigma = \{0, 1, 2, 3\}$. Is the language $L_1 = \{w \in \Sigma^* \mid$ in w, the number of 0's equals the number of 1's, and the number of 2's equals the number of 3's$\}$ context-free? Show your answer is correct.

2.1 Proof by Contradiction

The language

$$L_1 = \{w \in \{0, 1, 2, 3\}^* \mid \#_0(w) = \#_1(w) \text{ and } \#_2(w) = \#_3(w)\}$$

is **not** context-free. We prove this by contradiction using the pumping lemma for context-free languages.

Assumption: Assume, for contradiction, that L_1 is context-free. Then by the pumping lemma for context-free languages, there exists a pumping length $p > 0$ such that any string $w \in L_1$ with $|w| \geqslant p$ can be decomposed as $w = uvxyz$ where:

(i) $|vxy| \leqslant p$

(ii) $|vy| \geqslant 1$

(iii) For all $i \geqslant 0$, the string $uv^i xy^i z \in L_1$

Consider the string $w = 0^p 2^p 1^p 3^p$ which clearly lies in L_1 since it has exactly p zeros, p ones, p twos, and p threes, satisfying both conditions $\#_0(w) = \#_1(w)$ and $\#_2(w) = \#_3(w)$.

Analysis: By the pumping lemma, we can write $w = uvxyz$ with $|vxy| \leqslant p$ and $|vy| \geqslant 1$.
Since $|vxy| \leqslant p$, the substring vxy must be contained entirely within one of the four uniform blocks $0^p, 2^p, 1^p, 3^p$, or it must straddle exactly one adjacent boundary between these blocks.

We now show that in every possible case, pumping leads to a contradiction:

Case 1:
If vxy is contained within a single block, then v and y contain only one type of symbol (all 0s, all 2s, all 1s, or all 3s). Pumping up (setting $i = 2$) or pumping down (setting $i = 0$) changes that symbol's count while leaving at least one of its required partner counts untouched. For example, if vxy is contained in the 0^p block, then pumping increases or decreases the number of 0s without changing the number of 1s, breaking the requirement $\#_0 = \#_1$. Similar contradictions occur for the other blocks.

Case 2:
If vxy straddles the boundary between two blocks (say the 0/2 boundary), then pumping alters both symbols in the boundary but still leaves the symbols farther to the right unchanged. For instance, pumping across the 0/2 boundary changes the number of 0s without changing the number of 1s, breaking the requirement $\#_0 = \#_1$. Similar contradictions occur for the 2/1 and 1/3 boundaries.

Contradiction:
In all cases, no decomposition $uvxyz$ can satisfy the pumping lemma conditions while maintaining membership in L_1. This contradicts our assumption that L_1 is context-free. Therefore, L_1 is **not** a context-free language.

3 Assigned Reading [8 points]

Read the Introduction, Sections 1 and 2, and Section 9 up to the end of 9.I of Alan Turing's 1936 paper "On Computable Numbers, with an Application to the Entscheidungsproblem" that is available at: https://www.diochnos.com/about/TuringCompute.pdf.

(i) **[4 points]** Give an example of a computation that Turing would have probably believed a human would be unable to do without paper. Justify your answer.

(ii) **[4 points]** Is Turing more concerned with creating a machine that can simulate human computation or with creating a machine that humans can simulate? Justify your answer.

Please answer the two questions above briefly, referring to and/or quoting the text for support. A well thought-out paragraph should be sufficient for each question.

(i) Example of Computation

An example of a computation that Turing would have likely believed a human would be unable to do without paper is comparing two incredibly long numbers to confirm they are equal. In his paper, he uses the numbers 9999999999999999 and 999999999999999 to argue that the human mind has a finite number of states of mind, and this limited memory makes it difficult for a human to compute the quality of these numbers without paper. A human would be unable to tell at a glance if they are equal.

(ii) Turing's Concern

Turing is absolutely more concerned with creating a machine that can simulate human computation. Turing explicitly states "We may compare a man in the process of computing a real number to a machine", which tells us that the starting point for his rationale is a human, and the machines he is creating are a model of human computation. Furthermore, when we look at the machines tapes, Turing refers to them as the analogue of a human's state of mind. These ideas support the arguments that the simple operations of his machine include all those which are used in the computation of a number by a human. He is not interested in whether a human can trace the machine's steps, but whether the machine can successfully model the human's fundamental computational behavior.

4 Assigned Reading [8 points]

Read the article "Who Can Name the Bigger Number?" by Scott Aaronson, which you can find at: https://www.diochnos.com/teaching/CS3823/2025F/Aaronson99.pdf. This article may include concepts we have not yet covered, but the introduction to them should be sufficiently self-contained.

(i) **[4 points]** Going back to 5,000 B.C., name some advances in mathematics over time that have effectively allowed people to express bigger numbers than before the advent of those advances. You may do outside research to answer this question.[1] Include at least one advance from Aaronson's article.

(ii) **[4 points]** Would Aaronson say Turing machines are important even if they didn't have the immense real-world impact that they currently have? Why or why not?

(i) Mathematical Advances for Expressing Larger Numbers

Several key advances in mathematics have allowed people to express bigger numbers over time:

Exponents: Exponential notation was a major breakthrough because it allowed numbers to grow multiplicatively rather than just additively. For example, 10^3 is much larger than $10 + 10 + 10$. This concept became the foundation for expressing extremely large numbers in a concise way.

Ackermann Sequence: Aaronson discusses the Ackermann function, which is a systematic way to generate ever-larger numbers through a hierarchy of operations. It starts with addition, then multiplication, then exponentiation, then tetration, and continues beyond. This function grows faster than any primitive recursive function and shows how iterating operations on themselves can produce numbers far beyond what simple exponentiation can achieve.

Busy Beaver Numbers: Aaronson highlights the Busy Beaver function as one of the fastest-growing functions we can define. This function comes from computability theory and Turing machines, and it grows faster than any computable function, including the Ackermann function. The Busy Beaver function shows how the theory of computation lets us define numbers so large they exceed anything we could express with earlier mathematical methods.

(ii) Importance of Turing Machines Beyond Real-World Impact

Yes, Aaronson would argue that Turing machines are fundamentally important even without their real-world impact. He uses the hypothetical of explaining Turing machines to Archimedes at a time when computers were non-existent to illustrate this point. Aaronson writes that "big numbers have a way of imbuing abstract notions with reality," and he claims that the definition of science is "reason's attempt to compensate for our inability to perceive big numbers."

Turing machines allow us to rationalize and define larger numbers than ever before through their conceptual power. They provide the foundation for defining non-computable functions like the Busy Beaver, which grows faster than any computable function. This ability to formalize computability and explore the boundaries of what can be expressed mathematically makes Turing machines important to all of science, not just computer science. Even if they had never been physically built or used in practice, their theoretical importance in defining the limits of computation would make them a crucial tool for understanding what is and isn't possible in mathematics.

[1] A few examples will suffice; do not go overboard and write a research paper.

5 Designing a Turing Machine [8 points]

Let L_3 be the language $\{0^n 1^n \mid n \geqslant 1\}$ over $\Sigma = \{0, 1\}$.

(i) **[6 points]** Draw the state diagram for M_1, a deterministic Turing Machine that recognizes L_3. *It will be assumed that all the transitions not depicted go to the reject state, which may or may not be drawn.*

(ii) **[2 points]** Is M_1, as constructed in the previous part, a decider? (If it is hard to answer this question, you should perhaps change your answer to the previous question; that is, redesign the machine.)

(i) Turing Machine Diagram

Note: The square symbol □ used in JFLAP represents the blank symbol ⊔ in our formal notation.

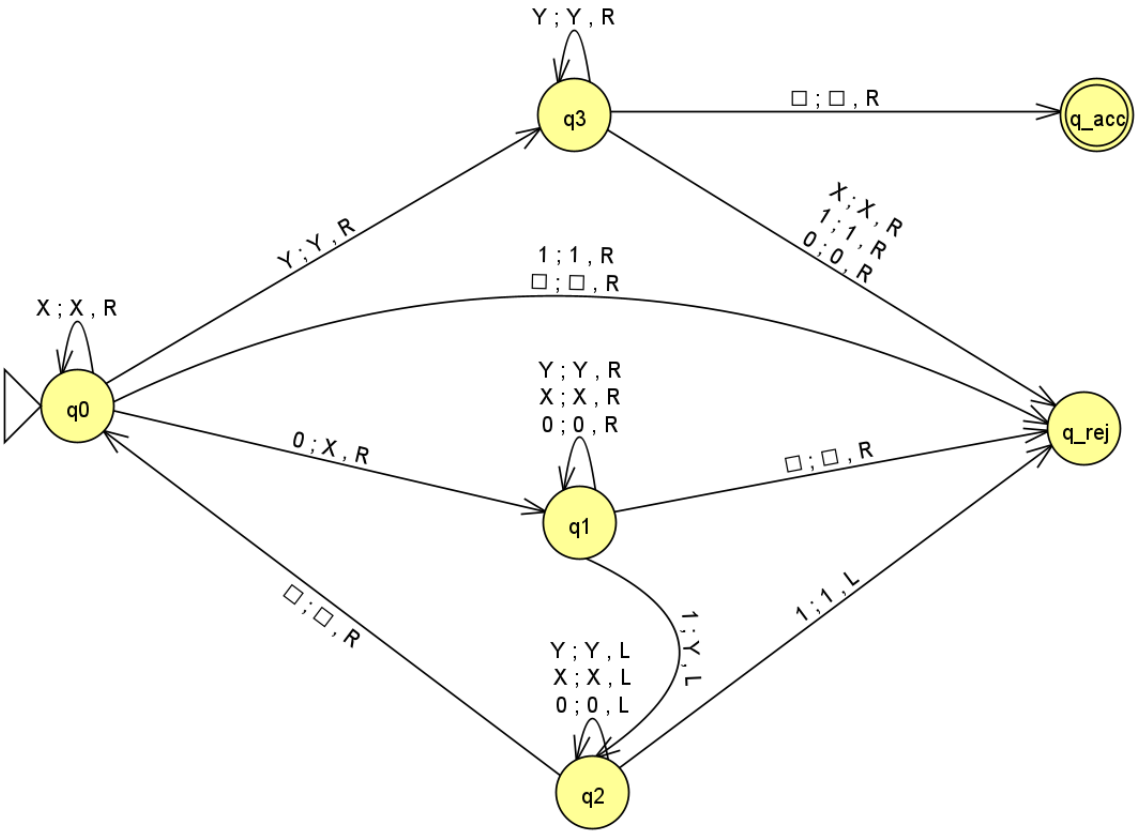

Figure 2: State diagram for M_1

(ii) Decider Proof

Yes, M_1 is a decider.

Proof that M_1 Always Halts:
A Turing machine is a decider if it:

1. Accepts all strings in the language
2. Rejects all strings not in the language
3. Always halts (never loops infinitely) on every input

Termination Analysis:
For any input string w:

- Each complete iteration ($q_0 \to q_1 \to q_2 \to q_0$) marks exactly one 0 and one 1
- Maximum possible iterations $= \min(\text{count}(0), \text{count}(1)) \leqslant |w|/2$
- Each iteration performs bounded work:
  - q_0: Scans through at most $|w|$ symbols
  - q_1: Scans through at most $|w|$ symbols
  - q_2: Scans through at most $|w|$ symbols
- Total operations $\leqslant |w|^3$ (polynomial bound)

Case Analysis:

1. $w \in L_3 = \{0^n 1^n \mid n \geqslant 1\}$**:**
   - Exactly n iterations, each marking one $(0, 1)$ pair
   - Ends in q_3, verifies only Y's remain, accepts
   - **Halts in accept state**
2. $w \notin L_3$**:**
   - **Wrong count ($0^n 1^m$, $n \neq m$):** Either runs out of 1's in q_1 ($n > m$) or finds unmarked 1's in q_3 ($n < m$)
   - **Wrong order:** Rejects immediately in q_0 (1 before 0) or q_2 (1 in 0's region)
   - **Mixed pattern:** Rejects in q_3 when finding $0/1$ during verification
   - **Empty string:** Rejects immediately in q_0
   - **Always halts in reject state**

No Infinite Loops Possible:

- The number of unmarked symbols strictly decreases each iteration
- Machine progresses monotonically through the input
- All transitions are well-defined for all symbols
- No cycles exist except the productive marking cycle

Conclusion:

Since M_1:

- **Accepts** all strings in L_3
- **Rejects** all strings not in L_3
- **Halts on every input** in finite time

M_1 **is a decider** and therefore L_3 **is Turing-decidable**.