# CS 3823 - Theory of Computation: Homework Assignment 3

Fall 2025 **Due:** Tuesday, November 4, 2025

---

**Related Reading.** Sections 3.3, 4.3, 5.1, 5.2, 6.1, 6.2, 7.1
**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.

---

**Student Information for the Solutions Submitted**

| | 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**

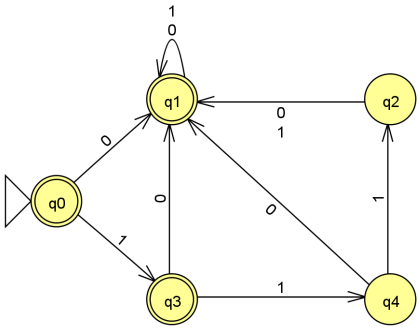
| Exercise | Pages | Your Score | Max |
|---|---|---|---|
| 1 | 2-3 | | 10 |
| 2 | 4 | | 5 |
| 3 | 5 | | 5 |
| 4 | 6-7 | | 10 |
| 5 | 8-9 | | 10 |
| **Total** | 2-10 | | 40 |

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

# 1 JFLAP [10 points]

Let the alphabet be $\Sigma = \{0, 1\}$ in every case below. Use JFLAP (http://www.jflap.org) to implement and test state machines. Include in each of your answers a **screenshot** that shows how each automaton looks like inside JFLAP, as well as **examples** of their execution **on some input strings**.

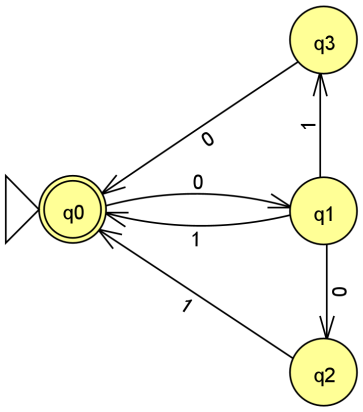
(i) **[5 pts]** Give a **DFA** recognizing the language $L_{1,a} = \{w \mid w \text{ is any string except 11 and 111}\}$.




| Input | Result |
|---|---|
| | Accept |
| 0 | Accept |
| 1 | Accept |
| 01 | Accept |
| 10 | Accept |
| 11 | Reject |
| 001 | Accept |
| 010 | Accept |
| 011 | Accept |
| 100 | Accept |
| 101 | Accept |
| 110 | Accept |
| 111 | Reject |
| 0101 | Accept |
| 1010 | Accept |
| 1110 | Accept |
| 0011 | Accept |
| 1100 | Accept |
| 1001 | Accept |
| 0001 | Accept |
| 0000 | Accept |
| 1111 | Accept |
| 1010101 | Accept |
| 1101010 | Accept |
| 0100101 10 | Accept |

Figure 1: DFA for $L_{1,a}$ implemented in JFLAP along with examples of its execution on some input strings

(ii) **[5 pts]** Give an **NFA** recognizing the language $L_{1,b} = (01 \cup 001 \cup 010)^*$.




| Input | Result |
|---|---|
| | Accept |
| 01 | Accept |
| 001 | Accept |
| 010 | Accept |
| 01 | Accept |
| 01001 | Accept |
| 01010 | Accept |
| 010001 | Accept |
| 01001 | Accept |
| 00101 | Accept |
| 001010 | Accept |
| 00101010 | Accept |
| 010101 | Accept |
| 01001001010 | Accept |
| 1 | Reject |
| 0 | Reject |
| 00 | Reject |
| 11 | Reject |
| 111 | Reject |
| 000 | Reject |
| 0100 | Reject |
| 1111 | Reject |
| 0000 | Reject |

Figure 2: NFA for $L_{1,b}$ implemented in JFLAP along with examples of its execution on some input strings

## 2 Beating a Finite Automaton in the Big Match [5 points]

This exercise is based on the paper *Beating A Finite Automaton in the Big Match*, by Lance Fortnow and Peter Kimmel. The paper is available at:

http://www.tark.org/proceedings/tark_jul22_98/p225-fortnow.pdf.

Please read the introduction and the preliminaries of this paper up to page 4 where the definition of "The Big Match" is given. Then jump to Section 3 and read Theorems 3.1 and 3.2 (with their proofs).

(i) **[3pts]** In the course we are using an idea similar to the one that is used in the proof of Theorem 3.1. What is that idea or theorem that we prove in class?

(ii) **[2pts]** Why does this idea fail to provide a positive result in Theorem 3.2?

**Solution:**

(i) **Idea from Theorem 3.1 similar to our course:**

The key idea in Theorem 3.1 is **state repetition** (or cycle detection) in finite automata.

**Proof strategy of Theorem 3.1:**

- P1 observes P2 (a DFA with $k$ states) for $3k$ rounds
- By the pigeonhole principle, P2 must enter a cycle
- Once in the cycle, P2's behavior becomes predictable and periodic
- P1 exploits this: plays $t$ when P2 is about to output $T$, or plays $h$ forever if the cycle only contains $H$

**Connection to our course:**

This is directly related to the **Pumping Lemma for Regular Languages**, which relies on the same fundamental property:

> *If a DFA has $k$ states and processes an input of length greater than $k$, then by the pigeonhole principle, it must visit some state twice. This creates a cycle that can be repeated ("pumped").*

Both Theorem 3.1 and the Pumping Lemma exploit the fact that:

- DFAs have **finite memory** (only $k$ states)
- After processing $k + 1$ inputs, a state must repeat
- This repetition creates predictable, periodic behavior

The Pumping Lemma uses this to show certain languages are non-regular, while Theorem 3.1 uses it to show that a player can exploit a finite automaton's predictable behavior.

(ii) **Why this idea fails in Theorem 3.2:**

In Theorem 3.2, P1 does **not know the number of states** $k$ of P2's DFA.

**The problem:**

The cycle detection method from Theorem 3.1 requires knowing $k$ to determine how long to observe (e.g., $3k$ rounds) to guarantee finding a cycle. Without knowing $k$, P1 faces a dilemma:

- **If P1 plays $t$ too early:** P2 might be designed to output $H$ at that specific round, giving P1 payoff 0
- **If P1 waits too long (or never plays $t$):** P2 could be designed to eventually output only $T$, but P1 never capitalizes on it, getting average payoff 0

**The adversarial construction:**

Since P1 is deterministic and doesn't know $k$, an adversary can construct a DFA with $k$ large enough that:

- The DFA outputs $H$ for longer than P1's observation period
- When P1 finally plays $t$ (at some predetermined round $n$), the DFA outputs $H$ at round $n$
- Or, if P1 never plays $t$, the DFA eventually outputs only $T$, but P1 misses all opportunities

**Conclusion:** Without knowing $k$, P1 cannot determine the observation length needed to guarantee seeing a complete cycle, so the DFA can always "hide" its periodic behavior beyond P1's detection threshold.

## 3 Regular Grammars [5 points]

Let $\Sigma = \{0, 1\}$. Find a regular grammar that generates the languange L shown below:

$$L = \{w \mid w \in \Sigma^* \text{ such that } w \text{ has at most two 0's}\} .$$

**Solution:**

We construct a right-linear grammar by creating states that count the number of 0's seen so far:

- State S: 0 zeros seen (start symbol)
- State A: 1 zero seen
- State B: 2 zeros seen

From each state, we can:

- Read a 1 and stay in the current state
- Read a 0 and move to the next state (unless we're already at 2 zeros)
- Terminate (accept), since all states represent valid strings

The regular grammar is:

$$\begin{aligned} S &\to 1S \mid 0A \mid \varepsilon \\ A &\to 1A \mid 0B \mid \varepsilon \\ B &\to 1B \mid \varepsilon \end{aligned}$$

This grammar generates all strings over $\{0, 1\}$ with at most two 0's. For example:

- String "01011": $S \Rightarrow 0A \Rightarrow 01A \Rightarrow 010B \Rightarrow 0101B \Rightarrow 01011B \Rightarrow 01011$ (2 zeros)
- String "000" cannot be generated: $S \Rightarrow 0A \Rightarrow 00B$, but B has no rule for producing another 0.

## 4 Context-Free Grammars [10 points]

Consider the context-free grammar $G_2$ shown below.

$$\begin{aligned} S &\longrightarrow 0X \\ X &\longrightarrow 0X \\ X &\longrightarrow 1X \\ X &\longrightarrow 1 \end{aligned}$$

(i) **[3 pts]** What language does $G_2$ recognize?

(ii) **[2 pts]** Is $L(G_2)$ regular? Why or why not?

(iii) **[3 pts]** Give a CFG in Chomsky Normal Form generating $L(G_2)$.

(iv) **[3 pts]** Is your CFG that you gave in the question above ambiguous? Why or why not?

**Solution:**

(i) **Language recognized by $G_2$:**

Starting with $S \Rightarrow 0X$, we observe that $X$ generates any string of 0's and 1's that *ends with 1*, since:

- The only way to terminate is via $X \to 1$
- Before terminating, we can use $X \to 0X$ or $X \to 1X$ any number of times

Therefore, $X$ generates $\{0,1\}^*1$, and:

$$L(G_2) = \{0w \mid w \in \{0,1\}^*, \text{ and } w \text{ ends with } 1\}$$

Equivalently: $L(G_2) = \{0u1 \mid u \in \{0,1\}^*\}$ (all binary strings starting with 0 and ending with 1).

(ii) **Is $L(G_2)$ regular?**

**Yes**, $L(G_2)$ is regular.

**Proof using Regular Grammar Definition:**

Recall that a **regular grammar** is one that is either right-linear or left-linear:

- **Right-linear:** All productions have the form $A \to xB$ or $A \to x$, where $A, B \in V$ (nonterminals) and $x \in T^*$ (string of terminals)
- **Left-linear:** All productions have the form $A \to Bx$ or $A \to x$

Let's examine the productions of $G_2$:

$$\begin{aligned} S &\to 0X \quad \text{(form: } A \to xB\text{, where } x = 0, B = X\text{)} \\ X &\to 0X \quad \text{(form: } A \to xB\text{, where } x = 0, B = X\text{)} \\ X &\to 1X \quad \text{(form: } A \to xB\text{, where } x = 1, B = X\text{)} \\ X &\to 1 \quad \text{(form: } A \to x\text{, where } x = 1\text{)} \end{aligned}$$

Every production is of the form $A \to xB$ or $A \to x$, so $G_2$ is a **right-linear grammar**.

Since $G_2$ is a regular grammar, it generates a regular language. Therefore, $L(G_2)$ is regular.

(iii) **CFG in Chomsky Normal Form:**

We convert the original grammar to CNF using the standard 5-step procedure:

**Original Grammar:**

$$\begin{aligned} S &\to 0X \\ X &\to 0X \mid 1X \mid 1 \end{aligned}$$

**Step 1: Add a new start variable.**

Not needed here, as $S$ doesn't appear on the right-hand side of any production.

**Step 2: Eliminate all $\lambda$-rules (of the form $A \to \lambda$).**

No $\lambda$-rules exist in this grammar, so no changes needed.

**Step 3: Eliminate all unit rules (of the form $A \to B$).**

No unit rules exist in this grammar, so no changes needed.

**Step 4: Patch up the grammar.**

The grammar still generates the same language $L(G_2) = \{0u1 \mid u \in \{0,1\}^*\}$, so no patching needed.

**Step 5: Convert the remaining rules into proper CNF form.**

CNF requires all productions to be either:

- $A \to BC$ (two nonterminals), or
- $A \to a$ (single terminal)

The problematic rules are $S \to 0X$, $X \to 0X$, and $X \to 1X$ (terminal mixed with nonterminal).

We introduce new nonterminals for each terminal:

- $T_0 \to 0$
- $T_1 \to 1$

Replace terminals in mixed productions:

- $S \to 0X$ becomes $S \to T_0X$
- $X \to 0X$ becomes $X \to T_0X$
- $X \to 1X$ becomes $X \to T_1X$
- $X \to 1$ stays as-is (already in CNF)

**Final CNF Grammar:**

$$\boxed{\begin{aligned} S &\to T_0X \\ X &\to T_0X \mid T_1X \mid 1 \\ T_0 &\to 0 \\ T_1 &\to 1 \end{aligned}}$$

All rules are now in proper CNF form.

(iv) **Is this CFG ambiguous?**

**No**, the grammar is **not ambiguous**.

**Proof using the Definition of Ambiguity:**

Recall that a grammar $G$ is **ambiguous** if it generates some string $w$ ambiguously, meaning that string has two or more different leftmost derivations.

Conversely, a grammar is **unambiguous** if every string in $L(G)$ has exactly one leftmost derivation.

For our grammar $G_2$, consider any string $w \in L(G_2)$. We know $w$ has the form $w = 0u1$ where $u \in \{0, 1\}^*$.

The leftmost derivation of $w$ must proceed as follows:

(a) **Start:** $S \Rightarrow 0X$ (this is the only production from $S$, so no choice here)

(b) **Middle:** For each symbol $s_i$ in $u = s_1 s_2 \dots s_k$:

- If $s_i = 0$: we must apply $X \Rightarrow 0X$ (uniquely determined by the symbol we need)
- If $s_i = 1$: we must apply $X \Rightarrow 1X$ (uniquely determined by the symbol we need)

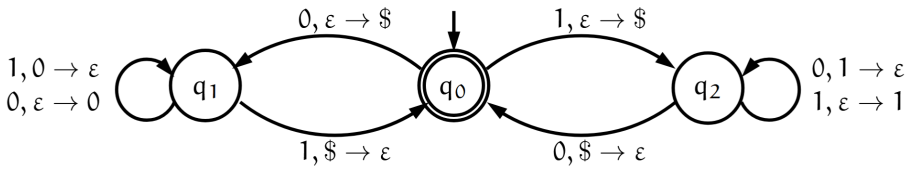
(c) **End:** $X \Rightarrow 1$ (this is the only way to terminate and produce the final 1)

Since each derivation step is uniquely determined by the target string, every string in $L(G_2)$ has **exactly one leftmost derivation**.

Therefore, by definition, the grammar $G_2$ is **unambiguous**.

# 5 Pushdown Automata [10 points]

Let $P_3$ be the PDA that is shown below.




(i) **[2 pts]** Give a formal description of $P_3$.

(ii) **[3 pts]** What language does $P_3$ recognize?

(iii) **[3 pts]** Give a context-free grammar that generates $L(P_3)$.

(iv) **[2 pts]** Is $L(P_3)$ regular? Why or why not?

**Solution:**

(i) **Formal description of $P_3$:**

**6-tuple Definition:**
$P_3 = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, \$\}$
- $q_0 \in Q$ is the start state
- $F = \{q_0\} \subseteq Q$

**Transition function:**
$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$
where $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$.

**Transition Table:**

| State | Input | Pop | New State | Push |
|---|---|---|---|---|
| $q_0$ | 0 | $\varepsilon$ | $q_1$ | \$ |
| $q_0$ | 1 | $\varepsilon$ | $q_2$ | \$ |
| $q_1$ | 0 | $\varepsilon$ | $q_1$ | 0 |
| $q_1$ | 1 | 0 | $q_1$ | $\varepsilon$ |
| $q_1$ | 1 | \$ | $q_0$ | $\varepsilon$ |
| $q_2$ | 0 | 1 | $q_2$ | $\varepsilon$ |
| $q_2$ | 1 | $\varepsilon$ | $q_2$ | 1 |
| $q_2$ | 0 | \$ | $q_0$ | $\varepsilon$ |

Note: Each row represents $\delta(\text{State}, \text{Input}, \text{Pop}) = \{(\text{New State}, \text{Push})\}$

(ii) **Language recognized by $P_3$:**

The PDA accepts strings based on the first symbol read:

**Case 1: Strings starting with 0**

When the first symbol is 0, the PDA enters state $q_1$. In $q_1$:

- Reading a 0 pushes it onto the stack
- Reading a 1 pops a 0 from the stack
- Reading a 1 when \$ is on top pops \$ and returns to $q_0$ (accept)

The PDA accepts when the stack becomes empty (reaching \$) after reading a 1. This happens when the string has exactly one more 1 than 0.

**Case 2: Strings starting with 1**

When the first symbol is 1, the PDA enters state $q_2$. In $q_2$:

- Reading a 1 pushes it onto the stack
- Reading a 0 pops a 1 from the stack
- Reading a 0 when \$ is on top pops \$ and returns to $q_0$ (accept)

The PDA accepts when the stack becomes empty (reaching \$) after reading a 0. This happens when the string has exactly one more 0 than 1.

**Language Definition:**

$L(P_3) = \{0w \mid w \in \{0,1\}^* \text{ and } w \text{ has exactly one more 1 than 0}\}$
$\cup \{1w \mid w \in \{0,1\}^* \text{ and } w \text{ has exactly one more 0 than 1}\}$

Equivalently: strings that start with 0 and have one more 1 than 0, or start with 1 and have one more 0 than 1.

(iii) **Context-free grammar for $L(P_3)$:**

We construct a CFG by considering the two cases separately:

**Grammar:**

$$\begin{aligned} S &\to 0E1 \mid 1E0 \\ E &\to 0E1E \mid 1E0E \mid \varepsilon \end{aligned}$$

**Explanation:**

The nonterminal E generates all strings with equal numbers of 0's and 1's (a balanced string):

- $E \to 0E1E$ adds a 0, recursively generates a balanced string, adds a 1, then recursively generates another balanced string
- $E \to 1E0E$ does the same but with 1 first, then 0
- $E \to \varepsilon$ allows termination

The start symbol S uses E to create strings with the desired imbalance:

- $S \to 0E1$: Start with 0, insert a balanced string (equal 0's and 1's), end with 1. This gives one more 1 than 0 overall.
- $S \to 1E0$: Start with 1, insert a balanced string (equal 0's and 1's), end with 0. This gives one more 0 than 1 overall.

For example, $S \Rightarrow 0E1 \Rightarrow 01$ gives string "01" (one 0, two 1's), and $S \Rightarrow 0E1 \Rightarrow 0(0E1E)1 \Rightarrow 0011$ gives string "0011" (two 0's, three 1's).

(iv) **Is $L(P_3)$ regular?**

**No**, $L(P_3)$ is **not regular**.

The language requires counting the number of 0's and 1's to ensure they differ by exactly one. This counting property cannot be achieved with a finite automaton, which has only finite memory.

More formally, we can prove this using the pumping lemma for regular languages: consider strings of the form $0^n1^{n+1}$ for large $n$. Any attempt to pump a substring will destroy the precise balance between 0's and 1's required by the language.