# CS 3823 - Theory of Computation: Homework Assignment 2

Fall 2025 **Due:** Friday, October 3, 2025

---

**Related Reading.** Chapter 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 | | 4 |
| 2 | 3-4 | | 8 |
| 3 | 5-6 | | 10 |
| 4 | 7 | | 4 |
| 5 | 8-9 | | 6 |
| 6 | 10-11 | | 8 |
| **Total** | 2-9 | | 40 |

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

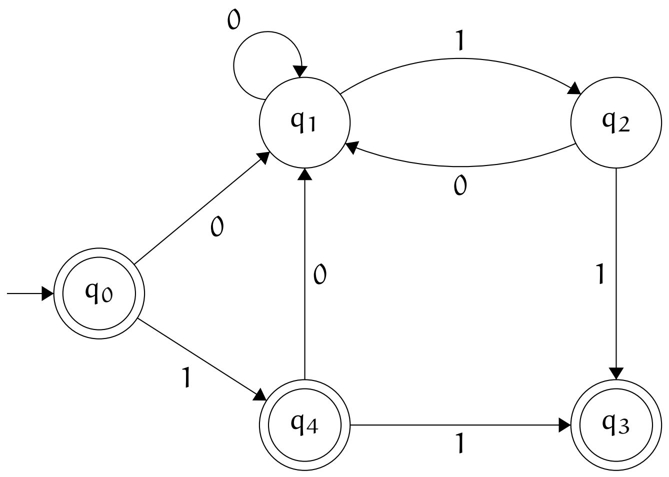
## 1 Design of NFAs [4 points; 2 points each]

Let $\Sigma = \{0, 1\}$. Give state diagrams for NFAs that recognize the following languages.

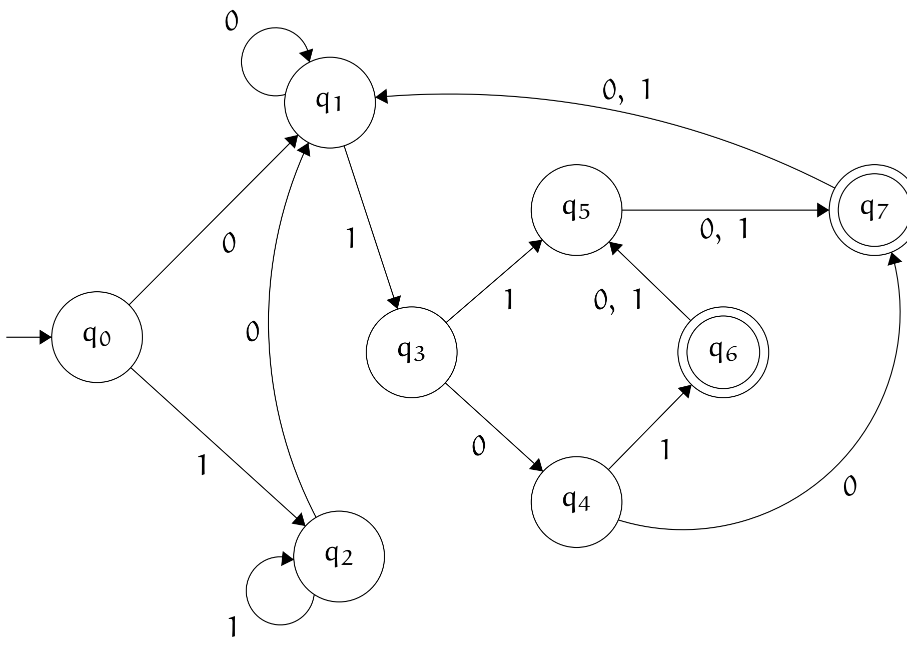
(i) $L_1 = \{w \mid w$ contains two consecutive 1s or $w$ contains no 0s$\}$.

(ii) $L_2 = \{w \mid w = w_1 w_2 \ldots w_n$ such that $w_{n-3} = 0$ and $w_{n-2} = 1\}$.

$L_1$ :



$L_2$ :



## 2 NFAs and DFAs [8 points]

Consider the NFA $N = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_1\})$, with $\delta$ as defined below.

| $\delta$ | $0$ | $1$ | $\varepsilon$ |
|---|---|---|---|
| $q_1$ | $\{q_1, q_2\}$ | $\emptyset$ | $\emptyset$ |
| $q_2$ | $\emptyset$ | $\{q_1\}$ | $\{q_1\}$ |

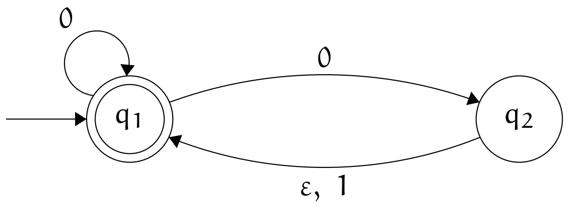
(i) **[2 points]** Draw the state diagram for $N$.

(ii) **[2 points]** What language does $N$ recognize?

(iii) **[3 points]** Let $M_1$ be a DFA recognizing $L(N)$. Using the 'power set' construction that we saw in class for Theorem 1.39 of the book, draw the state diagram for $M_1$ with the corresponding members of $\mathcal{P}(\{q_1, q_2\})$.

(iv) **[1 point]** Let $M_2$ be a DFA recognizing $L(M_1)$ but containing fewer states than $M_1$. Draw the state diagram of $M_2$.

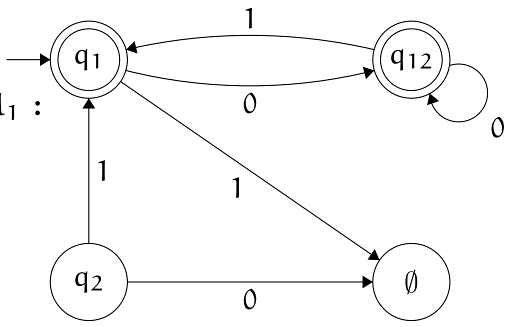
**(i)** $N$ **:**



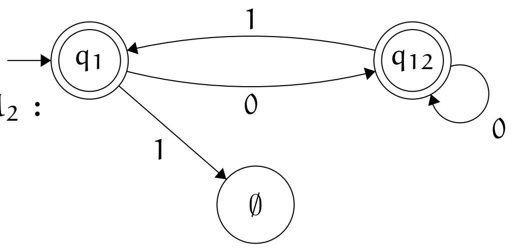
**(ii) Language** $N$ **recognizes:**

$$L(N) = \{w \in \{0,1\}^* \mid \text{every 1 in } w \text{ is immediately preceded by a 0}\} \text{ or } (0 \cup 01)^*$$

**(iii)** $M_1$ **:**



**(iv)** $M_2$ **:**



## 3 Representation [10 points]

(i) **[1 point]** Argue that if a language can be recognized by a DFA with $k$ states, then it can be recognized by an NFA with $k$ states.

Let $\Sigma = \{0, 1\}$. Let $\Sigma^n = \underbrace{\Sigma\Sigma \dots \Sigma}_{n}$. Consider the regular language $L_k = \Sigma^* 1 \Sigma^{k-1}$ for some positive integer $k$.

(ii) **[2 points]** Show that $L_k$ can be recognized by an NFA with $k + 1$ states.

(iii) **[4 points]** Prove that any DFA recognizing $L_k$ must have at least $2^k$ states.

(iv) **[2 points]** What does part (iii) of this question tell you about Theorem 1.39 from the book?

(v) **[1 point]** What do parts (i), (ii), and (iii) of this question tell you about DFAs as compared to NFAs?

**Answer to (i):**

Every DFA is, by definition, also an NFA. This is because an NFA (Nondeterministic Finite Automaton) is a generalization of a DFA (Deterministic Finite Automaton).

A DFA is simply a special case of an NFA where:

1. For each state and each input symbol, there is exactly one transition (no nondeterminism)
2. There are no $\varepsilon$-transitions (empty string transitions)

Therefore, any DFA with $k$ states can be viewed directly as an NFA with $k$ states that happens to be deterministic. The transition function, start state, and accepting states remain the same. No conversion is needed, the DFA already satisfies all the requirements of being an NFA.

**Answer to (ii):**

We can construct an NFA with $k + 1$ states as follows:

**States:** $Q = \{q_0, q_1, q_2, \dots, q_k\}$

**Construction:**

- $q_0$ is the start state
- $q_k$ is the only accepting state
- From $q_0$:
  - On input $0$ or $1$: stay in $q_0$ (loop)
  - On input $1$: *nondeterministically* transition to $q_1$ (this is the "guess" that we've seen the $1$ that is $k$ positions from the end)
- From $q_i$ (for $i = 1, 2, \dots, k - 1$):
  - On input $0$ or $1$: transition to $q_{i+1}$
- $q_k$ is the accepting state (reached after reading exactly $k - 1$ more symbols after the crucial $1$)

**How it works:** The NFA uses nondeterminism to "guess" when it sees the $1$ that is exactly $k$ positions from the end. It stays in $q_0$ reading arbitrary input until it nondeterministically decides to move to $q_1$ upon seeing a $1$. Then it must read exactly $k-1$ more characters (states $q_1 \to q_2 \to \cdots \to q_k$). If the string ends exactly when reaching $q_k$, it accepts.

**State count:** This NFA has exactly $k + 1$ states $(q_0, q_1, \dots, q_k)$.

**Answer to (iii):**

**Proof by contradiction:**

Assume, for the sake of contradiction, that there exists a DFA $M$ that recognizes $L_k$ and has fewer than $2^k$ states.

Consider all possible strings of length $k$ over $\Sigma = \{0, 1\}$. There are exactly $2^k$ such strings. Let's call this set $S = \Sigma^k = \{x_1, x_2, \dots, x_{2^k}\}$.

Since $M$ has fewer than $2^k$ states, and we are reading $2^k$ different strings, by the **Pigeonhole Principle**, at least two distinct strings from $S$ must lead to the same state in $M$.

Let's call these two strings $x$ and $y$ where $x \neq y$, and both are in $\Sigma^k$. Write:

- $x = x_1 x_2 \cdots x_k$
- $y = y_1 y_2 \cdots y_k$

Since $x \neq y$, there must exist some position $i$ where $1 \leqslant i \leqslant k$ and $x_i \neq y_i$.

Without loss of generality, assume $x_i = 1$ and $y_i = 0$.

Now consider the string $z$ consisting of any $(k - i)$ symbols from $\Sigma$. In other words, $z \in \Sigma^{k-i}$.

Observe:

- In $xz = x_1 x_2 \cdots x_k z$, the $k$-th character from the end is $x_i = 1$, so $xz \in L_k$
- In $yz = y_1 y_2 \cdots y_k z$, the $k$-th character from the end is $y_i = 0$, so $yz \notin L_k$

Since $x$ and $y$ lead to the same state in $M$, and $M$ is deterministic, reading the same suffix $z$ from that state must lead to the same final state. Therefore:

- Either both $xz$ and $yz$ are accepted by $M$
- Or both $xz$ and $yz$ are rejected by $M$

But we just showed that $xz \in L_k$ (should be accepted) and $yz \notin L_k$ (should be rejected).

This is a **contradiction.** The DFA $M$ cannot correctly recognize $L_k$ if it accepts both or rejects both.

Therefore, our assumption must be false, and any DFA recognizing $L_k$ must have **at least** $2^k$ **states**.

**Answer to (iv):**

Theorem 1.39 (the NFA to DFA conversion theorem) states that for every NFA, there exists an equivalent DFA. The standard subset construction algorithm can convert an NFA with $n$ states to a DFA with at most $2^n$ states.

Part (iii) demonstrates that this exponential blow-up is not just a theoretical upper bound, but can actually occur in practice:

- The NFA for $L_k$ has $k + 1$ states (from part ii)
- The minimum DFA for $L_k$ has $2^k$ states (from part iii)

This shows that:

1. The conversion from NFA to DFA can indeed require exponentially many states
2. The $2^n$ upper bound in the subset construction is tight, there exist languages where the exponential blow-up actually happens
3. While every NFA can be converted to a DFA (proving NFA and DFA have the same expressive power), the size cost can be exponential

**Answer to (v):**

Combining parts (i), (ii), and (iii):

**Expressive power:** DFAs and NFAs are equivalent in terms of the languages they can recognize (both recognize exactly the regular languages).

**Succinctness:** NFAs can be *exponentially more succinct* than DFAs. For the language $L_k$:

- NFA requires only $k + 1$ states
- DFA requires $2^k$ states

This is an exponential gap ($2^k$ vs. $k + 1$).

**Key insight:** While NFAs and DFAs recognize the same class of languages, NFAs can represent some languages much more efficiently. Nondeterminism allows for more compact representations at the cost of more complex acceptance semantics (multiple possible computation paths).

## 4 Regular Expressions [4 points; 2 points each]

Let $\Sigma = \{0, 1\}$. Give regular expressions for the following languages.

(i) $L_1 = \{w \mid \text{every even position of } w = w_1 w_2 \dots w_n \text{ has a } 0\}$.

(ii) $L_2 = \{w \mid w \text{ interpreted as a binary number is divisible by 2 (or 10 in binary)}\}$.

Note that $\varepsilon \in L_1$, whereas $\varepsilon \notin L_2$.

**Answer to (i):**

**Understanding:** Positions are numbered starting from 1. Even positions (2, 4, 6, ...) must be 0. Odd positions (1, 3, 5, ...) can be 0 or 1.

**Pattern:** The repeating pattern is $(0 \cup 1)0$ (any symbol followed by 0). This pattern repeats zero or more times, and we may have one final odd-positioned character (or none for the empty string).

**Regular expression:**

$$((0 \cup 1)0)^*(\varepsilon \cup 0 \cup 1)$$

**Verification:**

- $\varepsilon$: accepted (no even positions) ✓
- $0, 1$: accepted (only odd position 1) ✓
- $00, 10$: accepted (position 2 is 0) ✓
- $01, 11$: rejected (position 2 is 1) ×
- $1001$: accepted (positions 2 and 4 are both 0) ✓
- $1011$: rejected (position 4 is 1) ×

**Answer to (ii):**

**Key insight:** A binary number is divisible by 2 if and only if it ends in 0 (just like decimal numbers divisible by 10 end in 0).

**Regular expression:**

$$(0 \cup 1)^*0$$

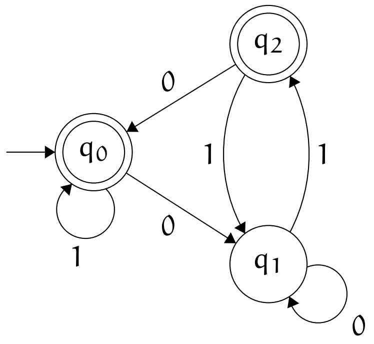
This accepts any string that ends in 0.

**Verification:**

- $0 = 0_{10}$: divisible by 2 ✓
- $10 = 2_{10}$: divisible by 2 ✓
- $100 = 4_{10}$: divisible by 2 ✓
- $110 = 6_{10}$: divisible by 2 ✓
- $1 = 1_{10}$: not divisible by 2 ×
- $11 = 3_{10}$: not divisible by 2 ×
- $\varepsilon$: not accepted ✓

## 5 DFAs to Regular Expressions [6 points]

Convert the following DFA to a regular expression.



Do so by starting from the equivalent GNFA, then remove the state $q_1$, then the state $q_2$ and finally the state $q_0$.

### Solution

**Step 0: Convert DFA to GNFA**

Add new start state $q_{start}$ and new accept state $q_{accept}$:

- $q_{start} \xrightarrow{\varepsilon} q_0$
- $q_0 \xrightarrow{\varepsilon} q_{accept}$ (since $q_0$ is accepting)
- $q_2 \xrightarrow{\varepsilon} q_{accept}$ (since $q_2$ is accepting)

All other transitions remain with their labels.

**Step 1: Remove $q_1$**

When removing a state $q$, for each pair of states $p$ and $r$ (where $p \neq q$ and $r \neq q$), replace the path $p \to q \to r$ with:

$$R_{new}(p, r) = R(p, r) \cup R(p, q) \cdot R(q, q)^* \cdot R(q, r)$$

**Paths through $q_1$:**

1. $q_0$ to $q_2$ through $q_1$: $q_0 \xrightarrow{0} q_1 \xrightarrow{0^*} q_1 \xrightarrow{1} q_2$

   New edge: $q_0 \xrightarrow{00^*1} q_2$

2. $q_2$ to $q_2$ through $q_1$: $q_2 \xrightarrow{1} q_1 \xrightarrow{0^*} q_1 \xrightarrow{1} q_2$

   New edge: $q_2 \xrightarrow{10^*1} q_2$ (self-loop)

**After removing $q_1$:**

- $q_0 \xrightarrow{1} q_0$ (self-loop)
- $q_0 \xrightarrow{00^*1} q_2$
- $q_0 \xrightarrow{\varepsilon} q_{accept}$
- $q_2 \xrightarrow{0} q_0$

- $q_2 \xrightarrow{10^*1} q_2$ (self-loop)
- $q_2 \xrightarrow{\varepsilon} q_{accept}$

**Step 2: Remove $q_2$**
**Paths through $q_2$:**

1. $q_0$ to $q_0$ through $q_2$: $q_0 \xrightarrow{00^*1} q_2 \xrightarrow{(10^*1)^*} q_2 \xrightarrow{0} q_0$

   New edge: $q_0 \xrightarrow{00^*1(10^*1)^*0} q_0$
2. $q_0$ to $q_{accept}$ through $q_2$: $q_0 \xrightarrow{00^*1} q_2 \xrightarrow{(10^*1)^*} q_2 \xrightarrow{\varepsilon} q_{accept}$

   New edge: $q_0 \xrightarrow{00^*1(10^*1)^*} q_{accept}$

**After removing $q_2$:**

- $q_0 \xrightarrow{1 \cup 00^*1(10^*1)^*0} q_0$ (combined self-loops)
- $q_0 \xrightarrow{\varepsilon \cup 00^*1(10^*1)^*} q_{accept}$

**Step 3: Remove $q_0$**
Apply the formula:

$$R(q_{start}, q_{accept}) = R(q_{start}, q_0) \cdot R(q_0, q_0)^* \cdot R(q_0, q_{accept})$$

Substituting:

$$= \varepsilon \cdot (1 \cup 00^*1(10^*1)^*0)^* \cdot (\varepsilon \cup 00^*1(10^*1)^*)$$

Since $\varepsilon$ is the identity for concatenation ($\varepsilon \cdot R = R$):

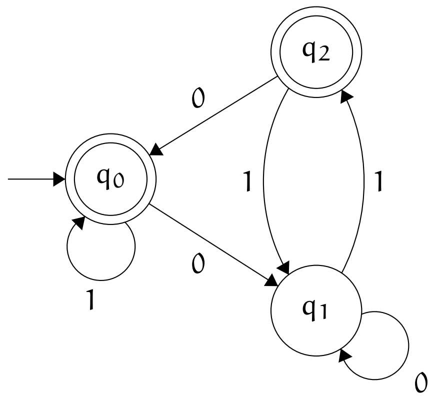
$$= (1 \cup 00^*1(10^*1)^*0)^* \cdot (\varepsilon \cup 00^*1(10^*1)^*)$$

**Final Regular Expression:**

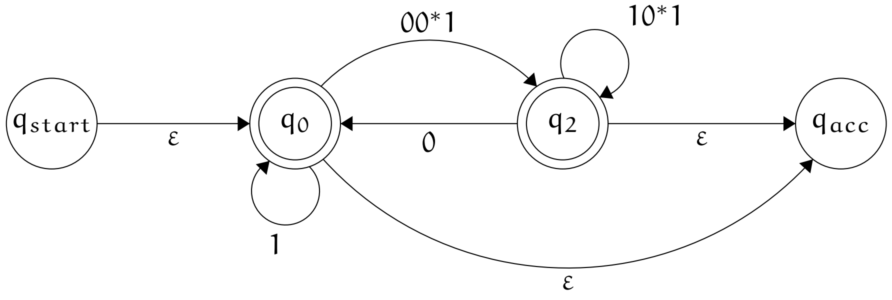
$$\boxed{(1 \cup 00^*1(10^*1)^*0)^*(\varepsilon \cup 00^*1(10^*1)^*)}$$

**Covnerting the DFA to a GNFA - diagrams:**

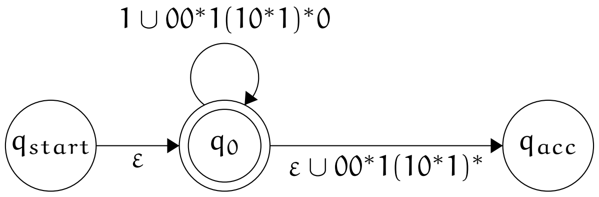
**DFA:**



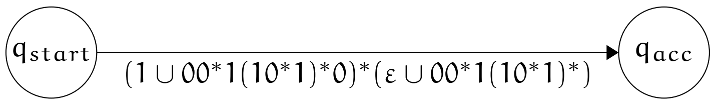
**Removing $q_1$ :**



**Removing $q_2$ :**



**Removing $q_0$ :**


$q_{start}$
$q_{acc}$
$(1 \cup 00^*1(10^*1)^*0)^*(\varepsilon \cup 00^*1(10^*1)^*)$

## 6 Non Regular Languages [8 points]

Use the Pumping Lemma to prove that the following languages are **not** regular.

(i) $L_1 = \{www \mid w \in \Sigma^*\}$, $\Sigma = \{0, 1\}$.

(ii) $L_2 = \left\{1^{2^n} \mid n \geqslant 1\right\}$, $\Sigma = \{1\}$.

### Solution

**(i) Proof for** $L_1 = \{www \mid w \in \Sigma^*\}$

**Proof by contradiction:**
Assume $L_1$ is regular. Then by the Pumping Lemma, there exists a pumping length $p$.
**Choose a string:** Let $w \in \Sigma^*$ be any string with $|w| = p$. Consider $s = www$.
Then $s \in L_1$ and $|s| = 3p \geqslant p$.
By the Pumping Lemma, we can write $s = xyz$ where:

- $|y| > 0$
- $|xy| \leqslant p$
- For all $i \geqslant 0$, $xy^iz \in L_1$

**Analysis of the split:**
Since $s = www$ with $|w| = p$, we can view the string as three consecutive copies of $w$:

$$s = \underbrace{w}_{\text{1st copy}}\ \underbrace{w}_{\text{2nd copy}}\ \underbrace{w}_{\text{3rd copy}}$$

The constraint $|xy| \leqslant p$ means that $xy$ is entirely contained within the first copy of $w$ (since the first $w$ has length $p$).
Since $xy$ lies within the first $w$:

- $x$ corresponds to some prefix of $w$ (possibly empty)
- $y$ corresponds to the next part of $w$ (non-empty, since $|y| > 0$)
- $z$ consists of the remainder of the first $w$ (if $|xy| < p$) plus the entire second and third copies of $w$, OR just the second and third copies of $w$ (if $|xy| = p$)

**Pumping to get a contradiction:**
Consider $i = 2$ (pump once). Then:

$$xy^2z = xyyz$$

Since $xy$ lies entirely within the first copy of $w$, pumping $y$ adds extra characters only to the first copy. The second and third copies of $w$ remain completely unchanged.
For $xy^2z$ to be in $L_1$, it must equal $w'w'w'$ for some string $w' \in \Sigma^*$ (three identical copies).
However:

- The first part of $xy^2z$ has been lengthened by $|y|$ characters (since we added an extra copy of $y$)
- The second and third parts remain as the original $w$

Since $|y| > 0$, the first part is now different from the second and third parts. Therefore, $xy^2z$ cannot be written as three identical copies.

Therefore, $xy^2z \notin L_1$.

This is a **contradiction.** Therefore, $L_1$ is not regular. □

**(ii) Proof for $L_2 = \left\{1^{2^n} \mid n \geqslant 1\right\}$**

**Proof by contradiction:**

Assume $L_2$ is regular. Then by the Pumping Lemma, there exists a pumping length $p$.

**Choose a string:** Let $s = 1^{2^p}$.

Then $s \in L_2$ and $|s| = 2^p \geqslant p$.

By the Pumping Lemma, we can write $s = xyz$ where:

- $|y| > 0$
- $|xy| \leqslant p$
- For all $i \geqslant 0$, $xy^iz \in L_2$

**Analysis of the constraints:**

From the Pumping Lemma conditions:

- Since $|xy| \leqslant p$, we have $|y| \leqslant p$
- Since $|y| > 0$, we know $y$ is non-empty

**Pumping to get a contradiction:**

Consider $i = 2$ (pump once). Compare the original string $xyz$ with the pumped string $xy^2z$.

The original string has length: $|xyz| = 2^p$

The pumped string has length: $|xy^2z| = |xyz| + |y| = 2^p + |y|$

Since $0 < |y| \leqslant p$, we can establish bounds on the pumped length:

- **Lower bound:** $|xy^2z| = 2^p + |y| > 2^p$ (since $|y| > 0$)
- **Upper bound:** $|xy^2z| = 2^p + |y| \leqslant 2^p + p$

Now, for $p \geqslant 1$, we have $p < 2^p$ (exponential growth), which means:

$$2^p + p < 2^p + 2^p = 2 \cdot 2^p = 2^{p+1}$$

Combining these inequalities:

$$2^p < |xy^2z| \leqslant 2^p + p < 2^{p+1}$$

**The contradiction:**

For $xy^2z$ to be in $L_2$, its length must be a power of 2. However, we've shown that:

$$2^p < |xy^2z| < 2^{p+1}$$

This means the length of $xy^2z$ is strictly between two consecutive powers of 2. Since there are no powers of 2 between $2^p$ and $2^{p+1}$, we have $xy^2z \notin L_2$.

This contradicts the Pumping Lemma, which states that $xy^2z$ must be in $L_2$.

Therefore, our assumption is false, and $L_2$ is not regular. □