LECTURE NOTES ON

# *CENG 491*

# *Formal Languages and Automata*

**PREPARED BY:** DR. EMRE SERMUTLU

**Based on the book:** Introduction to the Theory of Computation, Michael Sipser, 2nd ed.

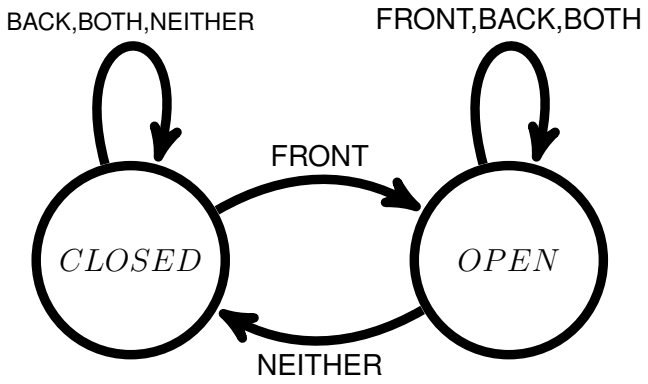
**LAST UPDATE:** NOVEMBER 27, 2015

# Week 1– Finite Automata

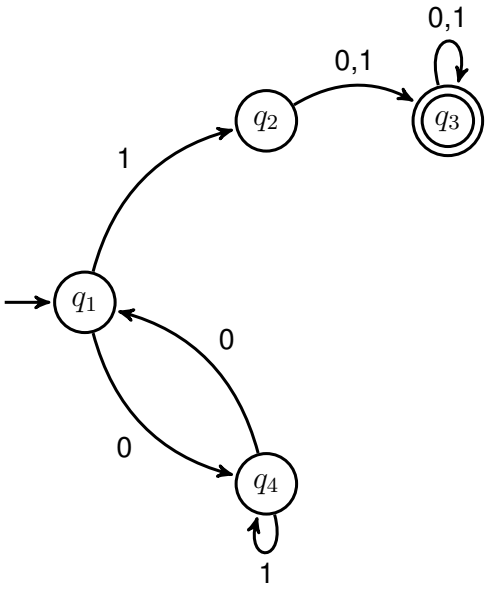
Consider an automatic door that opens when a person is in front and then closes. It should not open if someone is on the back side, or it may hit the person.
It has two states, OPEN and CLOSED and four possible inputs, so its state table looks like:

| | NEITHER | FRONT | BACK | BOTH |
|---|---|---|---|---|
| CLOSED | CLOSED | OPEN | CLOSED | CLOSED |
| OPEN | CLOSED | OPEN | OPEN | OPEN |

We can express this table using the following diagram:




The state diagram of a typicial finite automaton looks like this:




An alphabet is a finite set of symbols. A string over an alphabet is a finite sequence of symbols. A string that has zero length is called the empty string and denoted by $\varepsilon$. A language is a set of

strings.
Here, the alphabet is: $\Sigma = \{0, 1\}$. The states of the automaton are:$q_1, q_2, q_3$ and $q_4$. The start state is $q_1$ and the accept state is $q_3$. The arrows are called transitions. Note that there is one and only one arrow for each state and each symbol in the alphabet.
The input to this automaton is a string over the alphabet, such as $010001$. The output is accept or reject.

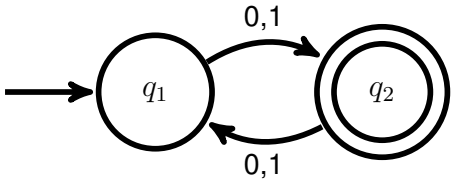
**The Formal Definition of Finite Automaton**

A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

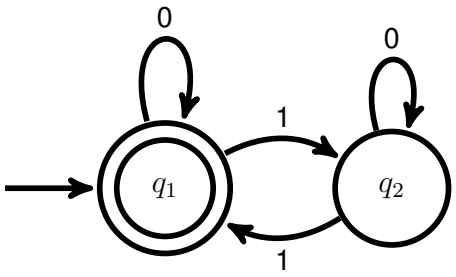
- $Q$ is a finite set called the states,
- $\Sigma$ is a finite set called the alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $q_o \in Q$ is the start state,
- $F \subseteq Q$ is the set of accept states.

If $A$ is the set of all strings that machine $M$ accepts, we say that $A$ is the language of machine $M$ and write $L(M) = A$. This means, $M$ recognizes $A$.

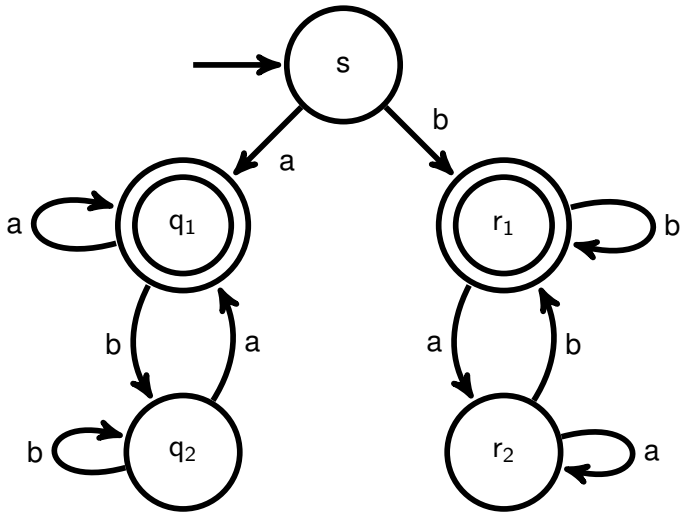
**Exercise 1-1:** What language does the following automaton recognize?




**Exercise 1-2:** What language does the following automaton recognize?



**Exercise 1-3:** What language does the following automaton recognize?




**Exercise 1-4:** Find a finite automaton that recognizes the following languages over the alphabet $\Sigma = \{0, 1\}$:

**a)** $A = \{w \mid$ the last two digits are 1$\}$

**b)** $B = \{w \mid$ w contains 111$\}$

**c)** $C = \{w \mid$ the number of digits of w is zero modulo 3$\}$

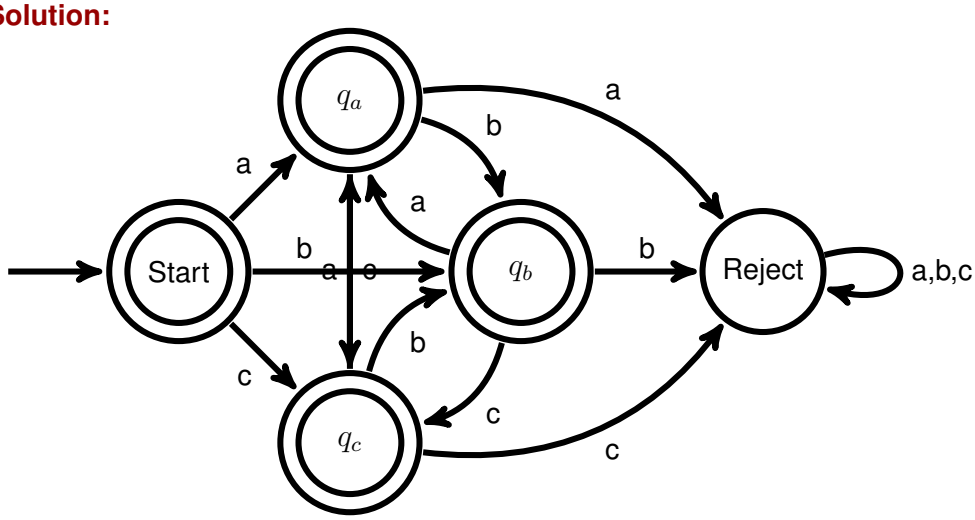
**d)** $D = \{w \mid$ w does not contain 101$\}$

**e)** $E = \{w \mid$ the length of w is at most 4$\}$

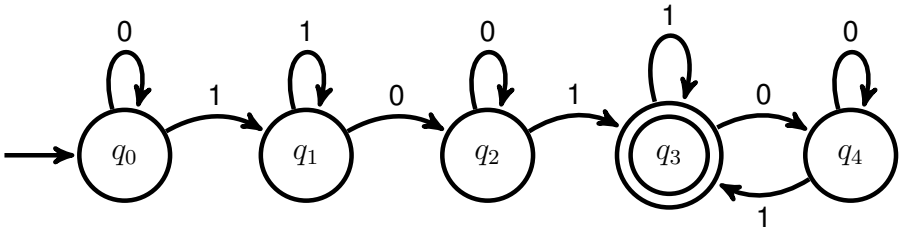
**f)** $F = \{w \mid$ the third position from the end is 1$\}$

**Exercise 1-5:** Design a DFA for all strings over the alphabet $\Sigma = \{a, b, c\}$ in which there is no $aa$, no $bb$ and no $cc$.

**Solution:**



**Exercise 1-6:** What language does the following DFA recognize? Describe verbally.



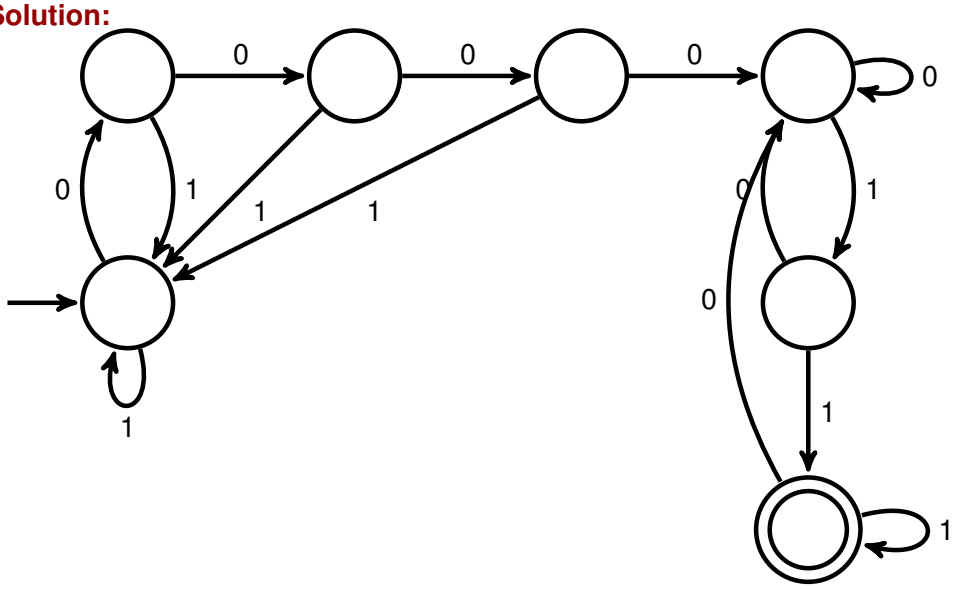

**Solution:**
Many answers are possible. Some of them, starting with the simplest are:

- Contains $10$ and ends with $1$.
- Contains $10$ and $01$ and ends with $1$.
- Contains at least two symbol changes and ends with $1$.
- Contains at least 3 symbols, $1, 0$ and $1$ in this order (but not necessarily consecutively) and ends with $1$.

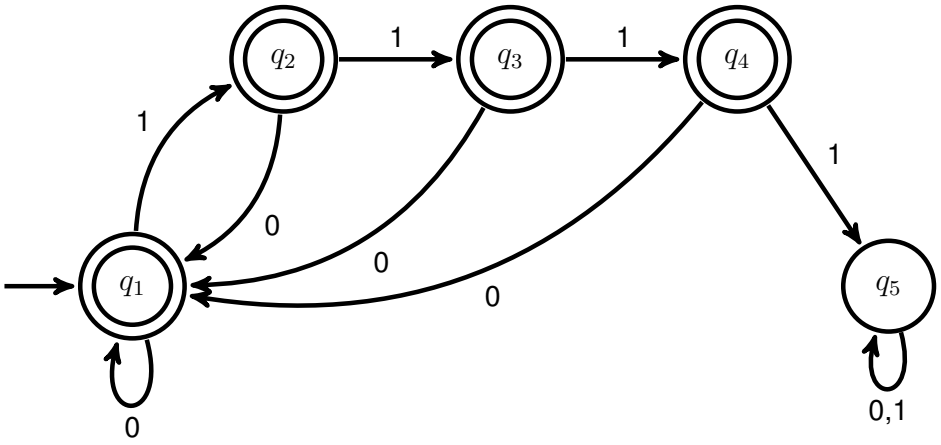
**Exercise 1-7:** Give the state diagram of a DFA that recognizes the language $A$ over alphabet $\Sigma = \{0, 1\}$ where

$$A = \{w \mid w \text{ contains } 0000 \text{ and ends with } 11\}$$

**Solution:**



**Exercise 1-8:** The following DFA recognizes the language $B$ over alphabet $\Sigma = \{0, 1\}$. Describe $B$ verbally.

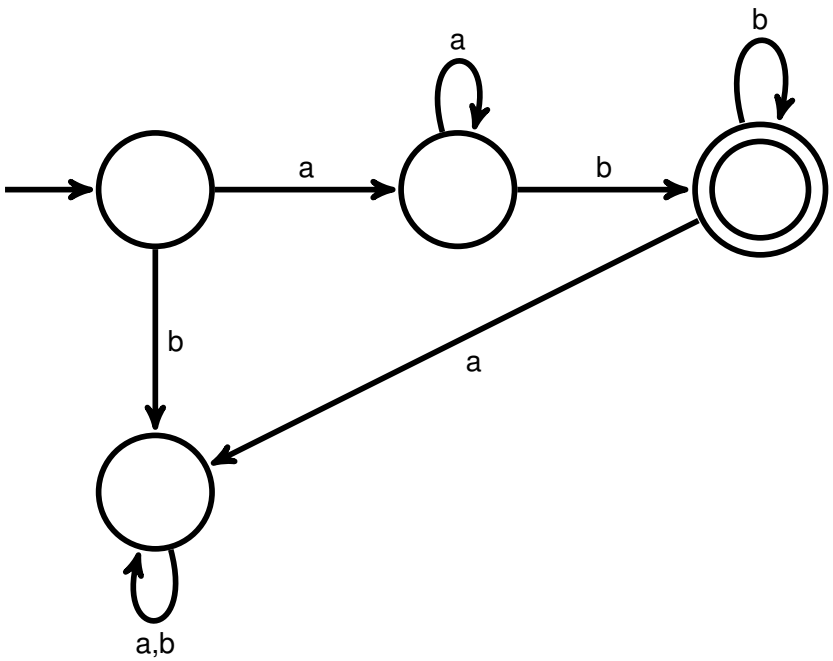



**Solution:**
Does NOT contain 1111.

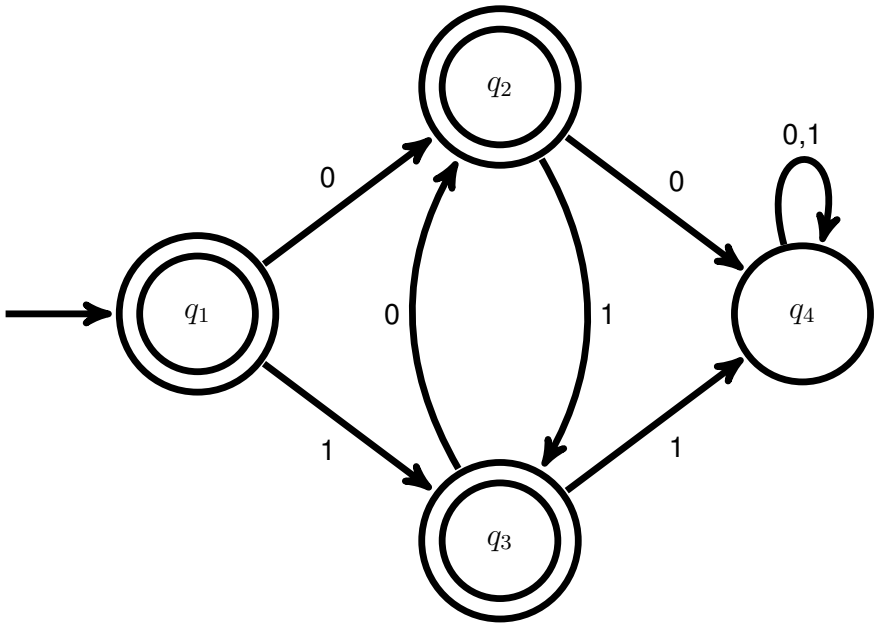
**Exercise 1-9:** Give the state diagram of a DFA that recognizes the language $A$ over alphabet $\Sigma = \{a, b\}$ where

$$A = \{w \mid w = a^n b^m, \quad n, m \geqslant 1\}$$

**Solution:**



**Exercise 1-10:** The following automaton recognizes the language $L$. Describe $L$ verbally.

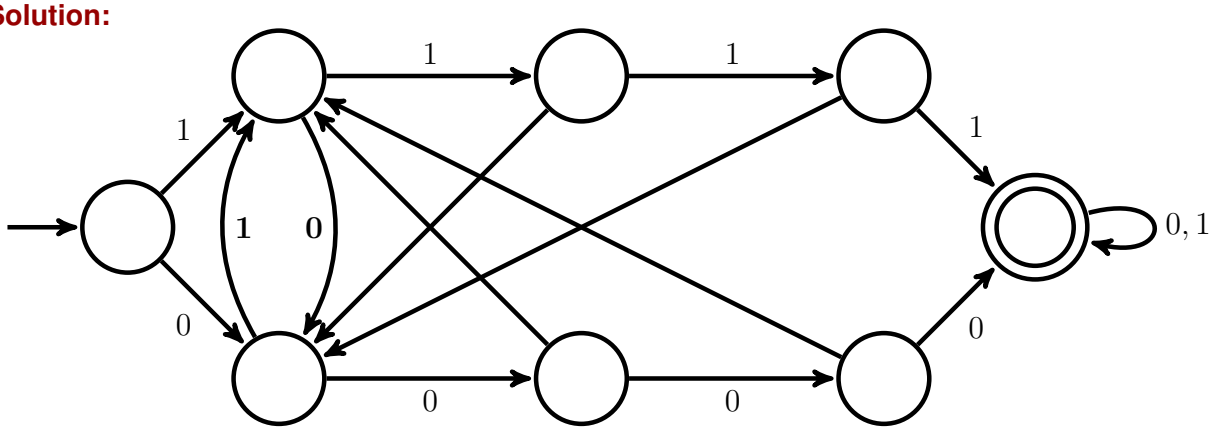



**Solution:**
The binary strings that do not contain 00 or 11.

**Exercise 1-11:** Give the state diagram of a DFA that recognizes the language $A$ over alphabet $\Sigma = \{0, 1\}$ where

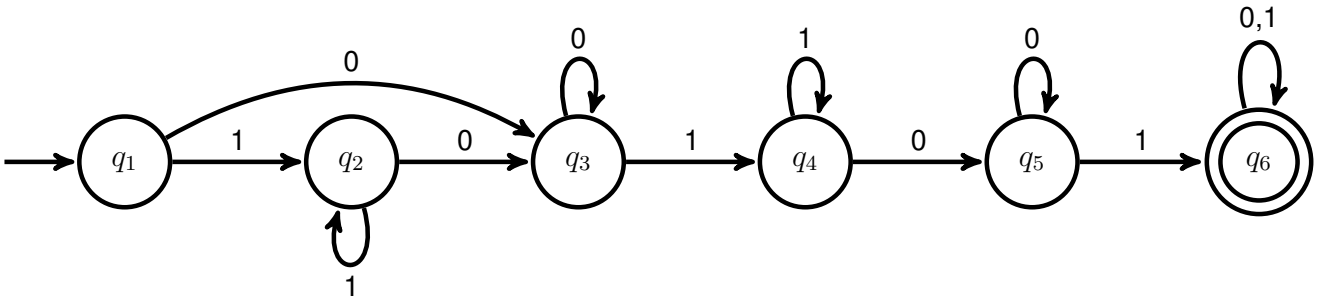
$$A = \{w \mid w \text{ contains } 1111 \text{ or } 0000\}$$

**Solution:**




**Exercise 1-12:** The following DFA recognizes the language $B$ over alphabet $\Sigma = \{0, 1\}$.

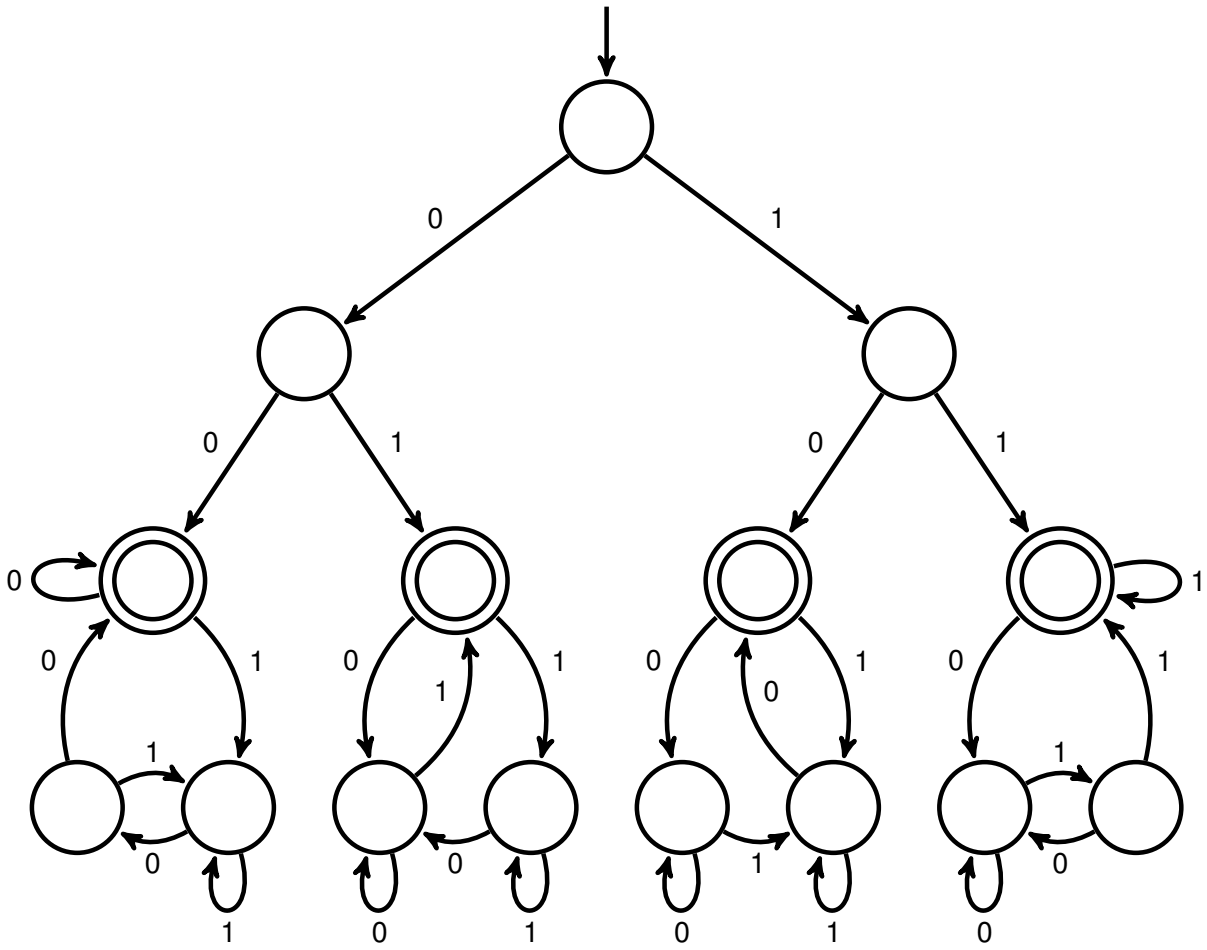
Describe $B$ verbally.




**Solution:** Contains at least two 01's
OR
If it starts with zero, changes symbols at least 3 times, if it starts with 1, changes symbols at least 4 times.

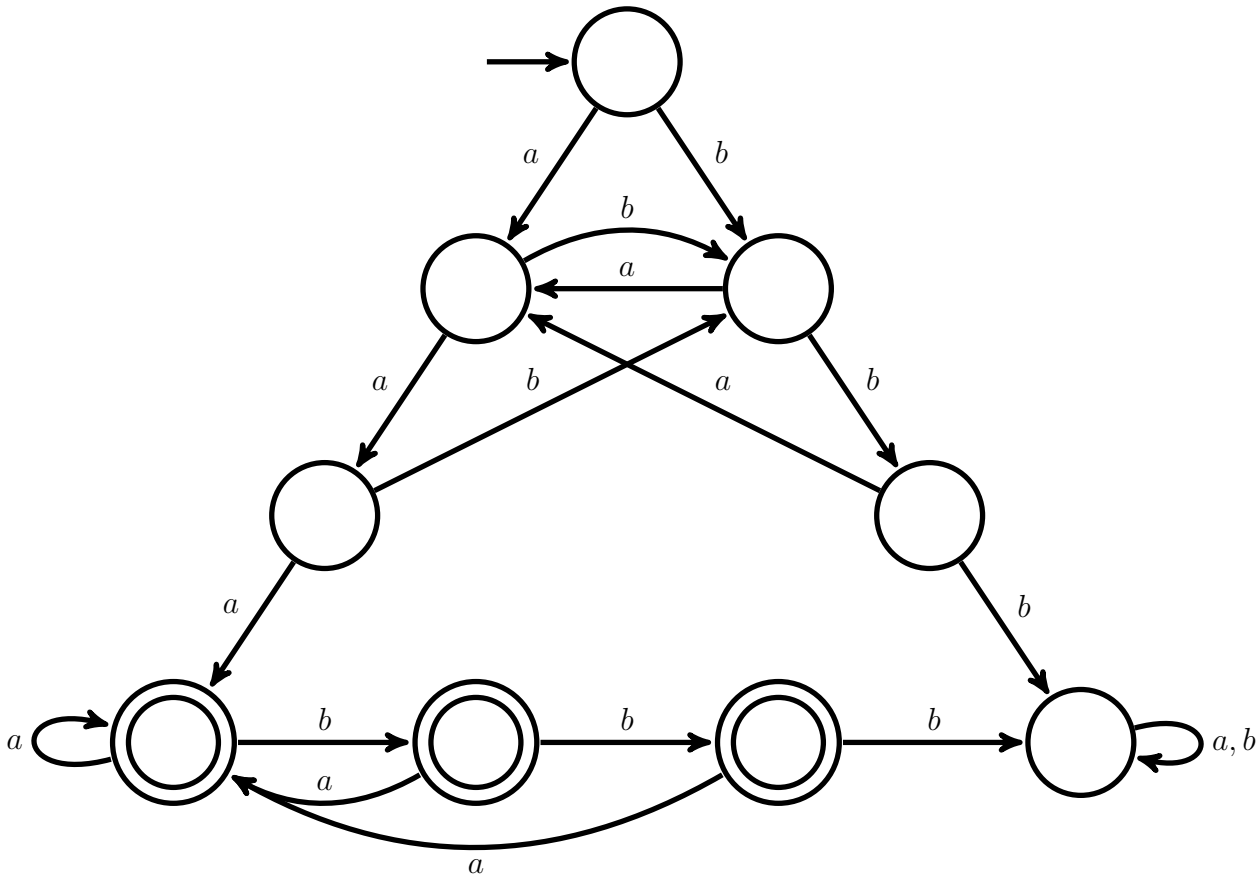
**Exercise 1-13:** Give the state diagram of a DFA that recognizes the language $A$ over alphabet $\Sigma = \{0, 1\}$ where $|w| \geqslant 2$ and the first two and the last two digits of $w$ are identical.
For example:
$10010010 \in A, 0011100 \in A$ but
$001 \notin A, 100001 \notin A$

**Solution:**



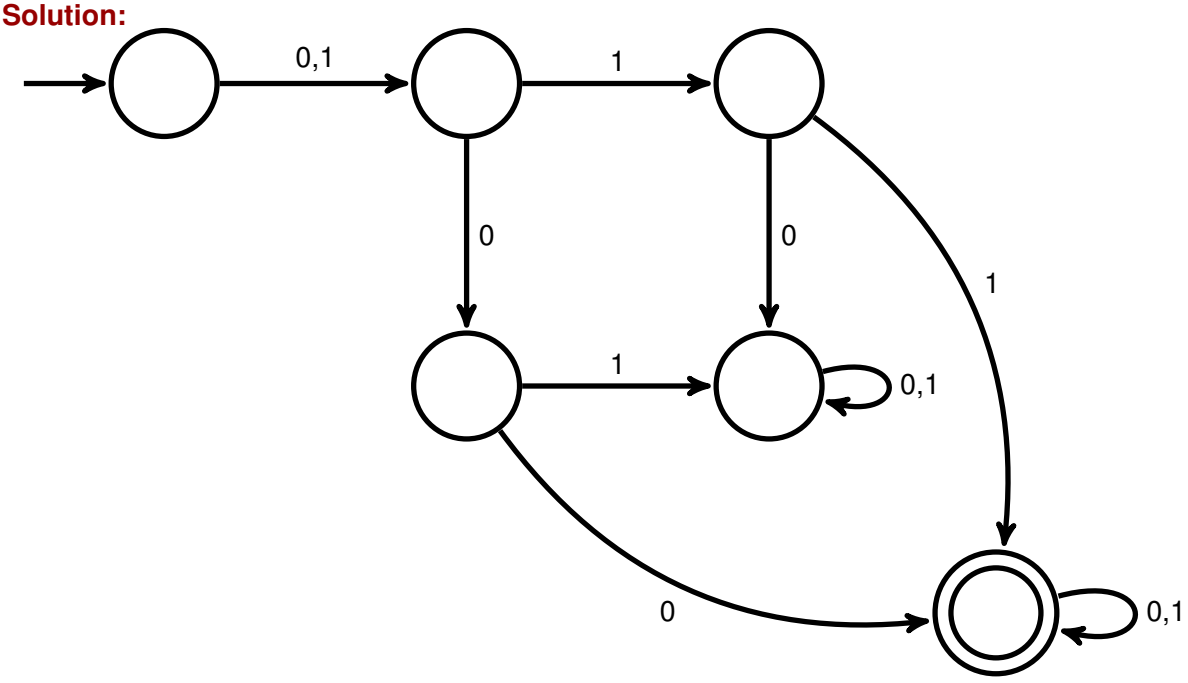

**Exercise 1-14:**
Give the state diagram of a DFA that recognizes the language $A$ over alphabet $\Sigma = \{a, b\}$ where $A = \{w \mid w$ contains $aaa$ but does not contain $bbb\}$.

**Solution:**




**Exercise 1-15:**
Let:
$A = \{$Strings of length at least 3 whose second and third items are the same$\}$
over alphabet $\Sigma = \{0, 1\}$.
Give the state diagram of the DFA that recognizes this language.

**Solution:**



# Week 2– Regular Operations

**Regular Languages:** A language is called a regular language if some finite automaton recognizes it. Note that a DFA accepts a string and recognizes a language.

**Regular Operations:** Let $A$ and $B$ be languages. We define the regular operations *union*, *concatenation*, and *star* as follows:

- Union: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
- Star: $A^* = \{x_1 x_2 \dots x_k \mid k \geqslant 0 \text{ and each } x_i \in A\}$.

Note that $k = 0$ is a possibility therefore $\varepsilon \in A^*$ whatever $A$ is.

**Example:** Consider two languages $A = \{CENG, ECE, IE\}$ and $B = \{STUDENT, TEACHER\}$. Then,

$A \cup B = \{$ CENG,ECE,IE, STUDENT,TEACHER$\}$
$A \circ B = \{$ CENGSTUDENT,CENGTEACHER, ECESTUDENT,$\dots\}$
$A^* = \{\varepsilon,$ CENG,ECE,IE,CENGCENG,CENGECE,CENGIE,$\dots\}$

**Theorem:** The class of regular languages is closed under the union operation. In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \cup A_2$.

**Proof:** During computation, the automaton $M$ must keep in mind that the given string might be from $A_1$ or $A_2$. So if the machines $M_1$ and $M_2$ have $k_1$ and $k_2$ states, $M$ must have $k = k_1 k_2$ states.

**Theorem:** The class of regular languages is closed under the concatenation operation. In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \circ A_2$.
To prove this theorem, we have to break the given string at all possible points.

# Week 3– Nondeterminism

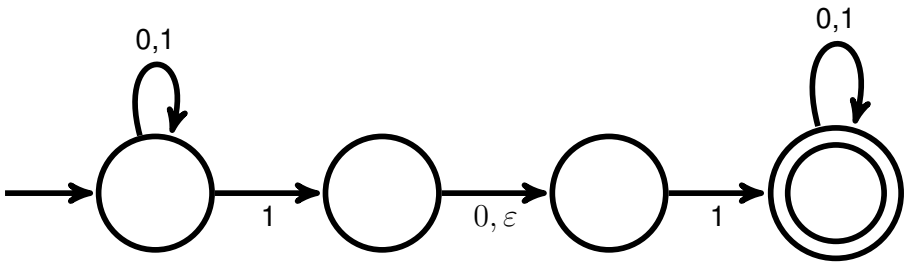
Nondeterminism is a kind of parallel computation. We will use the abbreviation DFA for deterministic and NFA for nondeterministic finite automata.
In an NFA, arrows may be labeled $\varepsilon$. A state may have 0,1 or many arrows leaving for each symbol. If there are many arrows, the machine splits at that point. If there are no arrows, the machine dies.
If any one machine accepts, the NFA accepts.

**Formal Definition of an NFA:**

A nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- $Q$ is a finite set of states
- $\Sigma$ is a finite alphabet
- $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

For example




**Theorem:** Every nondeterministic finite automaton has an equivalent deterministic finite automaton.
Sketch of Proof: Given an NFA with n states, construct a DFA with $2^n$ states. These correspond to all possible subsets of states.
**Corollary:** A language is regular if and only if some nondeterministic finite automaton recognizes it.

**Theorem:** The class of regular languages is closed under the union operation
New Proof Idea:Use an NFA that combines two DFA's in parallel with $\varepsilon$ arrows.

**Theorem:** The class of regular languages is closed under the concatenation operation.
Proof Idea:Use an NFA that combines two DFA's in series with $\varepsilon$ arrows

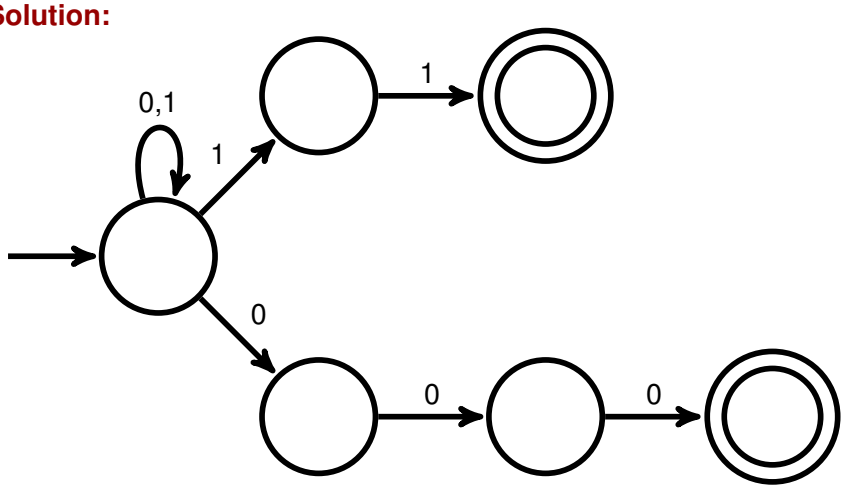
**Theorem:** The class of regular languages is closed under the star operation.

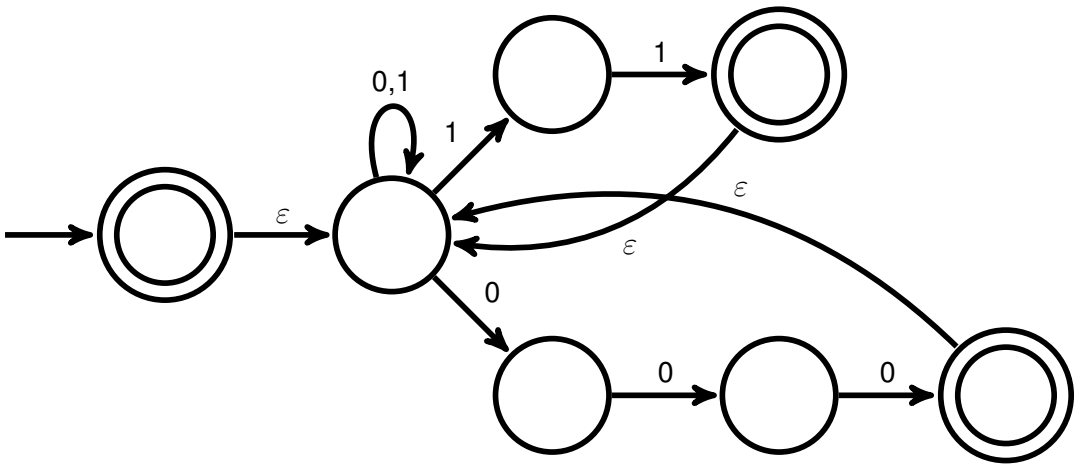
**Exercise 3-1:** Let $A$ be a language over $\Sigma = \{0, 1\}$. $A = \{w \mid w$ ends with 11 or 000$\}$.

**a)** Find an NFA that recognizes $A$.

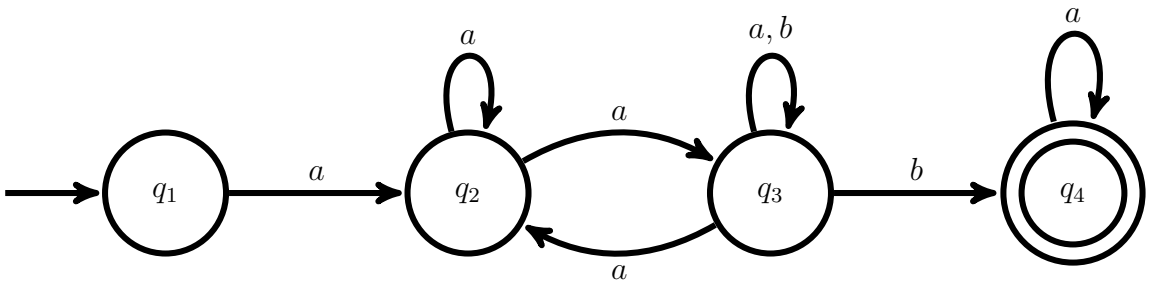
**b)** Find an NFA that recognizes $A^*$.

**Solution:**

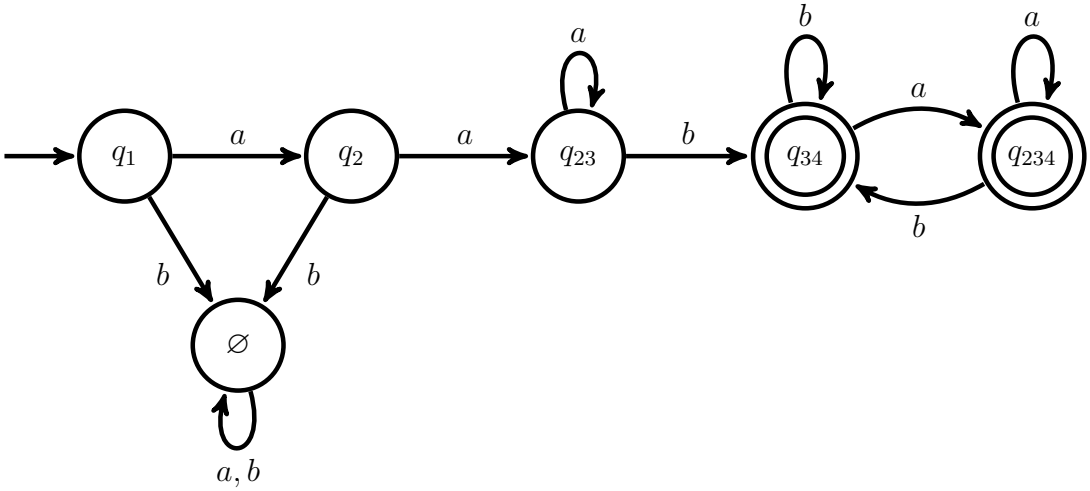


**Exercise 3-2:** Find an equivalent DFA for the following NFA:




**Solution:**

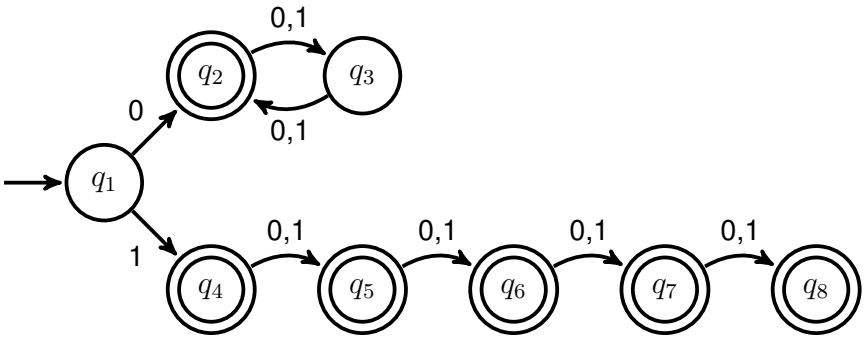


**Exercise 3-3:** Give the state diagram of a DFA or an NFA that recognizes the language $A$ over alphabet $\Sigma = \{0, 1\}$ where

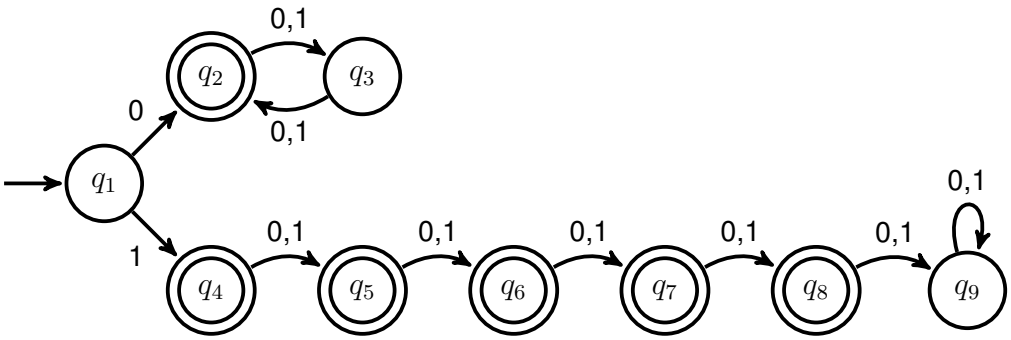
$$A = \{w \mid w \text{ starts with 0 and has odd length, or starts with 1 and has length at most 5}\}$$

**Solution:**

NFA:




DFA:

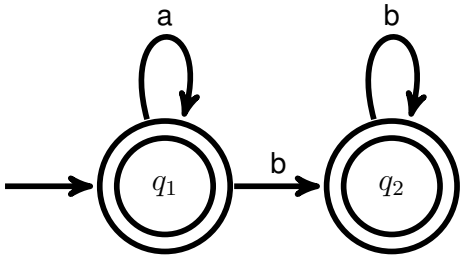


**Exercise 3-4:** Give the state diagram of a DFA or an NFA that recognizes the language $A$ over alphabet $\Sigma = \{a, b\}$ where

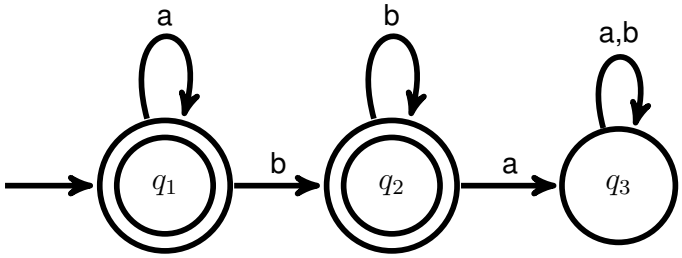
$$A = \{w \mid w = a^*b^*\}$$

**Solution:**

NFA:




DFA:



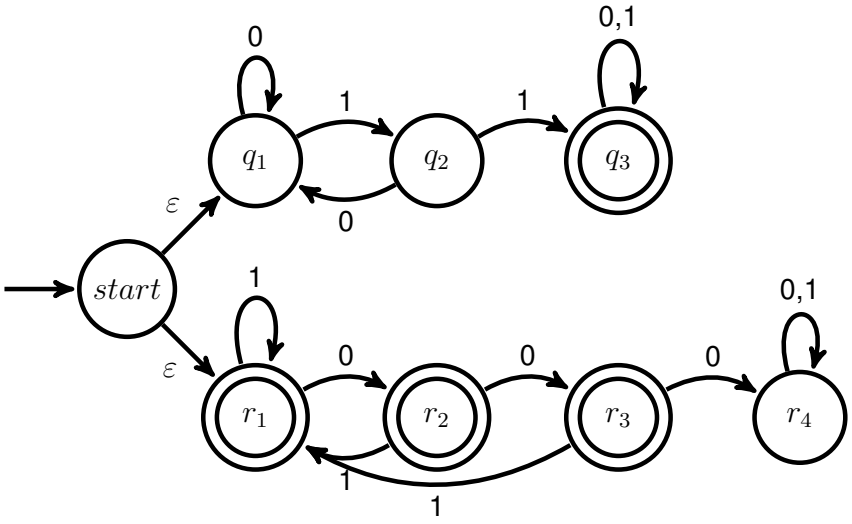
**Exercise 3-5:**
Let $A$ and $B$ be languages over $\Sigma = \{0, 1\}$.

$$A = \{w \mid w \text{ contains } 11\}$$

$$B = \{w \mid w \text{ does not contain } 000\}$$

Give the state diagram of an NFA that recognizes $A \cup B$.

**Solution:**



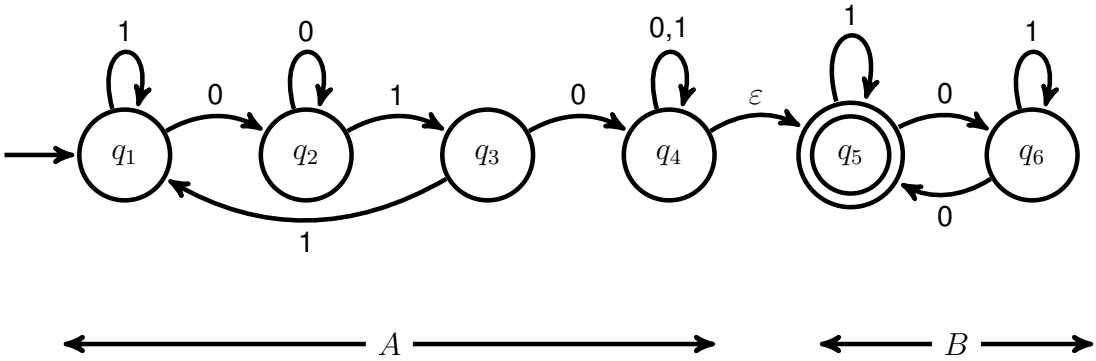

**Exercise 3-6:**
Let $A$ and $B$ be languages over $\Sigma = \{0, 1\}$.

$$A = \{w \mid w \text{ contains } 010\}$$

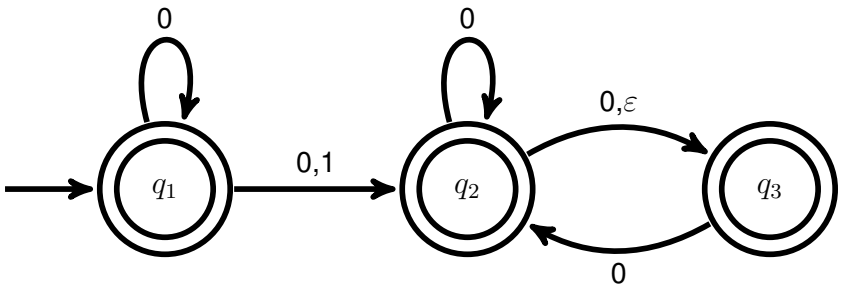
$$B = \{w \mid w \text{ contains even number of zeros}\}$$

Give the state diagram of an NFA that recognizes $A \circ B$.

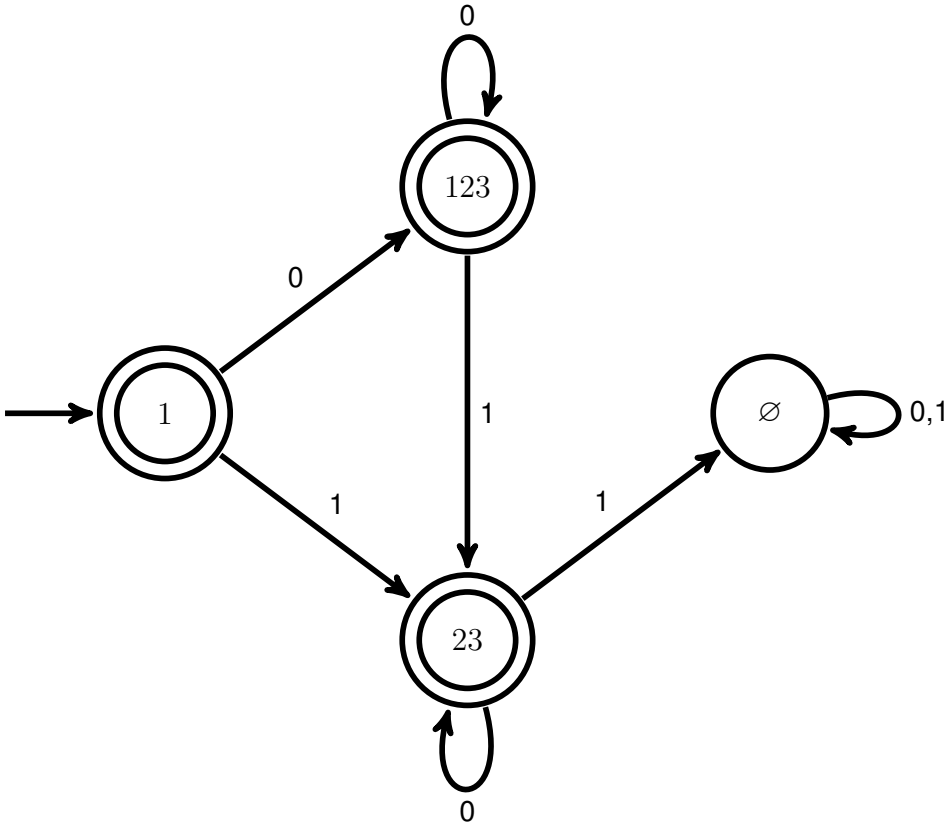
**Solution:**



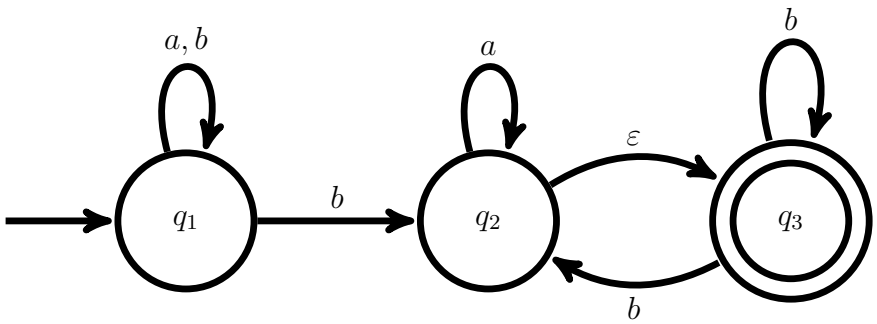
**Exercise 3-7:** Convert the following NFA to a DFA:



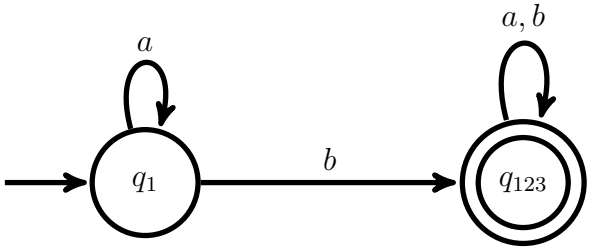

**Solution:**




**Exercise 3-8:** Convert the following NFA to a DFA:




**Solution:**


a
a, b
b
$q_1$
$q_{123}$

# Week 4– Regular Expressions

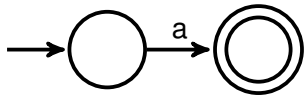
We say that $R$ is a regular expression if $R$ is

- $a$ for some $a$ in the alphabet $\Sigma$.
- $\varepsilon$
- $\varnothing$
- $R_1 \cup R_2$ where $R_1$ and $R_2$ are regular expressions
- $R_1 \circ R_2$ where $R_1$ and $R_2$ are regular expressions
- $R_1^*$ where $R_1$ is a regular expression.

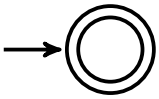
**Example:** Assume $\Sigma = \{0, 1\}$
$0^*10^*$: Contains a single 1
$\Sigma^*1\Sigma^*$ Has at least one 1
$(\Sigma\Sigma)^*$: Strings of even length
$(0 \cup \varepsilon)1^* = 01^* \cup 1^*$
$\varnothing^* = \varepsilon$

**Theorem:** A language is regular if and only if some regular expression describes it. **Proof - Part 1:** Given a regular expression, find an NFA that recognizes it.

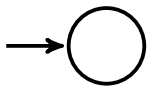
$1) R = a$ :




$2) R = \varepsilon$ :

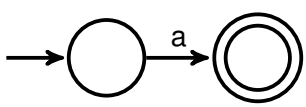


$3) R = \varnothing$ :

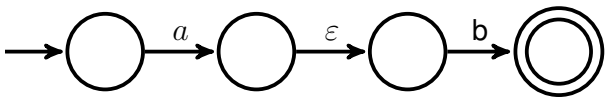


The cases for union, concatenation and star operations were covered before.
**Example:** Convert the regular expression $(a \cup ab)^*$ into an NFA.

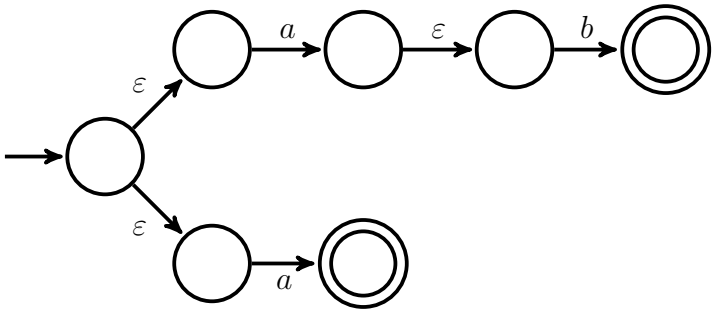
$a$ :



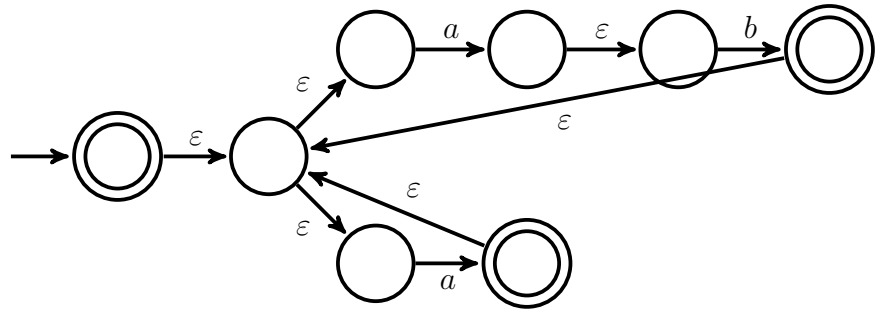
$ab$ :



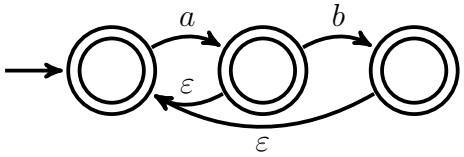

$a \cup ab$ :



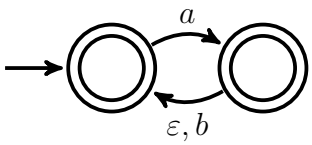

$(a \cup ab)^*$ :




This procedure gives an automaton that works, but it is not the simplest one. Another solution is:




An even simpler solution is:

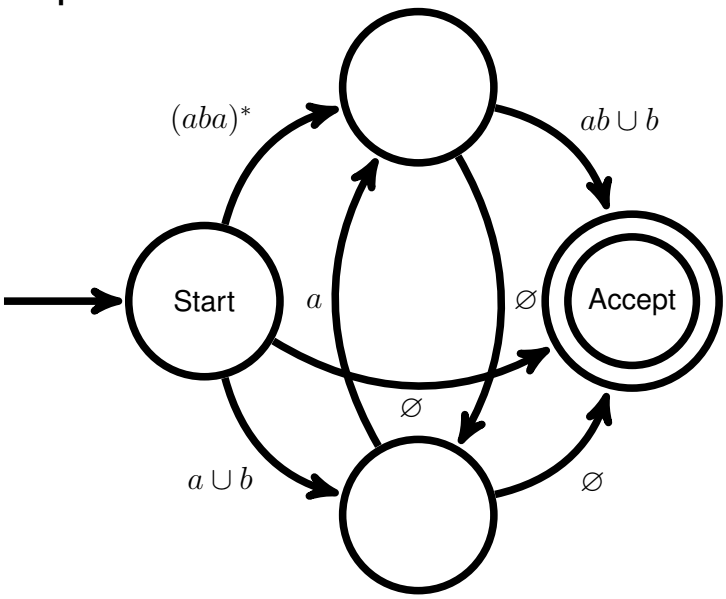



**Proof - Part 2:** Given an NFA, find a regular expression equivalent to its language.

We will need a new concept for this part.
A generalized non-deterministic finite automaton (GNFA) has the following properties:

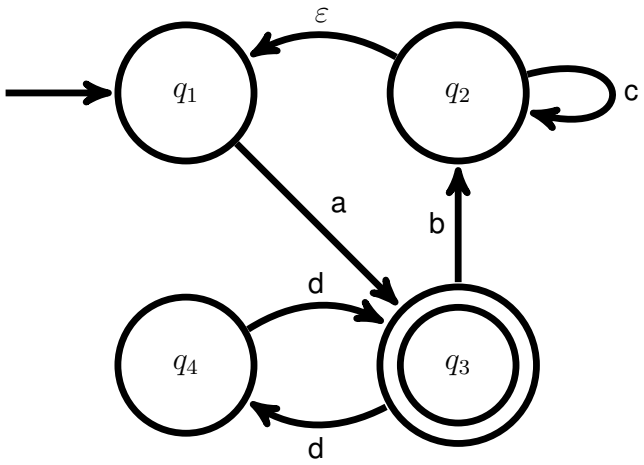
- Arrow labels are regular expressions.
- There is a single accept state, distinct from start state.
- Arrows do not enter the start state and they do not leave the accept state.
- Each state is connected to each other (except for start and accept). If necessary, we use $\varnothing$ as label.

**Example:**



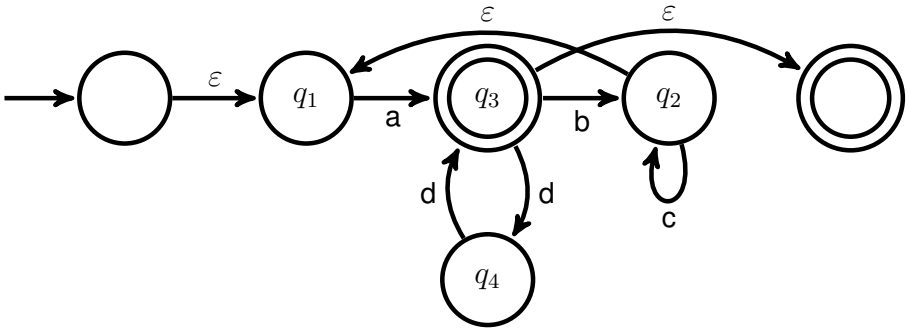

We can convert an NFA into GNFA by adding new start and accept states, connecting these to old ones with $\varepsilon$ arrows, merging labels of arrows and adding arrows with $\varnothing$.
Now we can eliminate all states of GNFA except start and accept states one by one until we obtain a 2-state machine. The label on the arrow will give us the regular expression we are looking for.

**Exercise 4-1:** Find the regular expression equivalent to the following NFA:

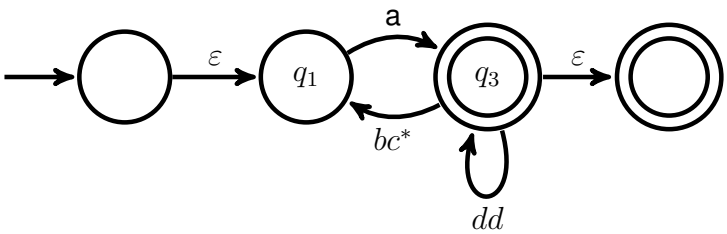



**Solution:**

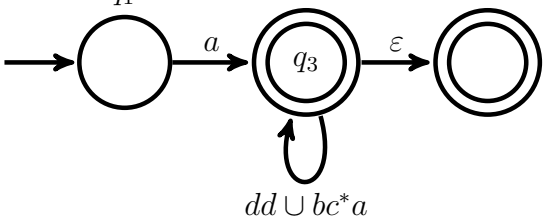
GNFA:



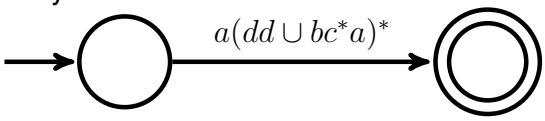

Eliminate $q_4$ and $q_2$:




Eliminate $q_1$:




Finally:




So the answer is: $a(dd \cup bc^*a)^*$

Other possible answers are: $[a(dd)^*bc^*]^*a(dd)^*$ or $a[(dd)^*(bc^*a)*]^*$

**Exercise 4-2:**

Let $\Sigma = \{a, b\}$. A language is described by the regular expression: $(\Sigma^* a \Sigma^* a \cup bb) b^*$

**a)** Give a string that is in this language.

**b)** Give a string that is NOT in this language.

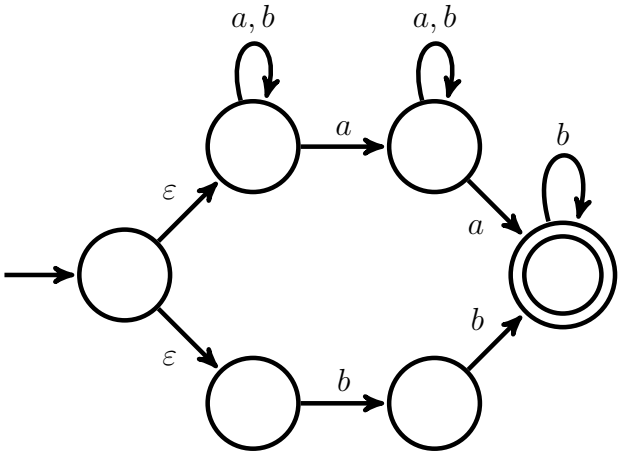
**c)** Give an NFA that recognizes this language.

**Solution:**

**a)** $aa$ or $bb$ or $aabbbbb$ or $bbababab$

**b)** $a$ or $b$ or $bbbba$

**c)**



**Exercise 4-3:**
Let $\Sigma = \{0, 1\}$. A language is described by the regular expression: $(\varepsilon \cup 000 \cup 11)(01)^*$

**a)** Give a string that is in this language.

**b)** Give a string that is NOT in this language.

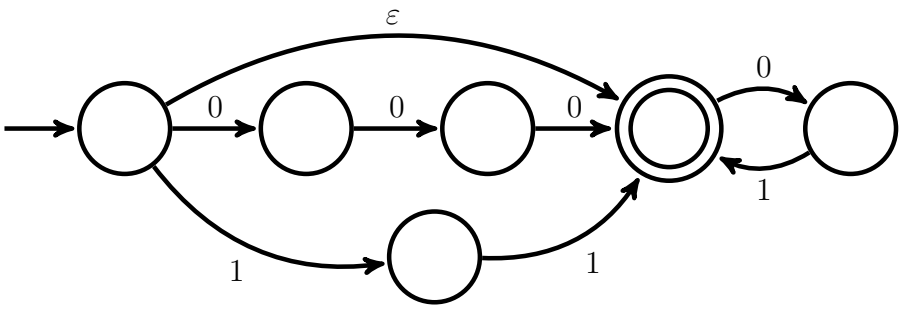
**c)** Give an NFA that recognizes this language.

**Solution:**

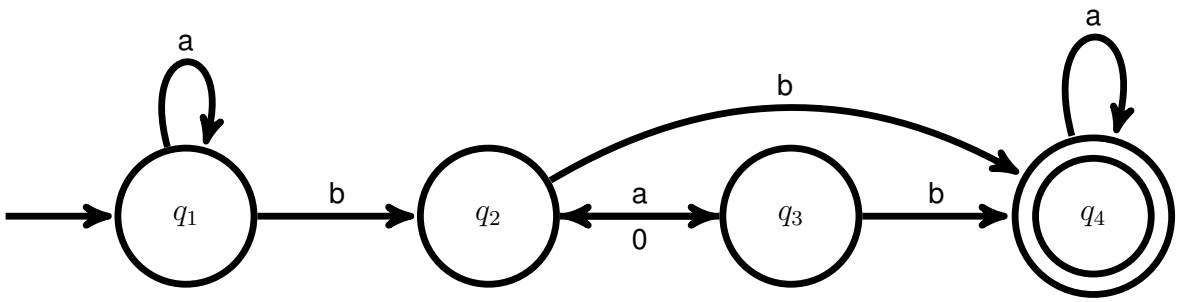
**a)** $000$ or $11$ or $00001$ or $010101$

**b)** $11111$ or $0000$ or $0$ or $1$.

**c)**




**Exercise 4-4:** Find a regular expression equivalent to the language recognized by the following NFA:

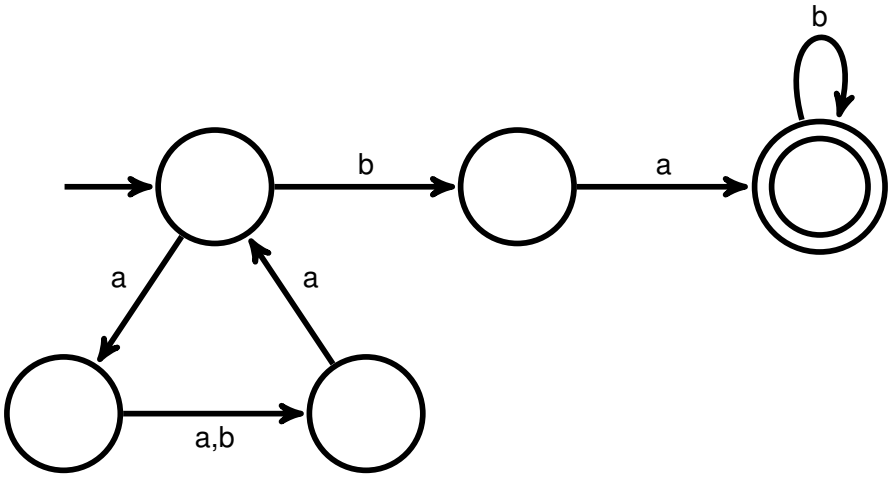



**Solution:**

$$a^*b(b \cup ab)a^*$$

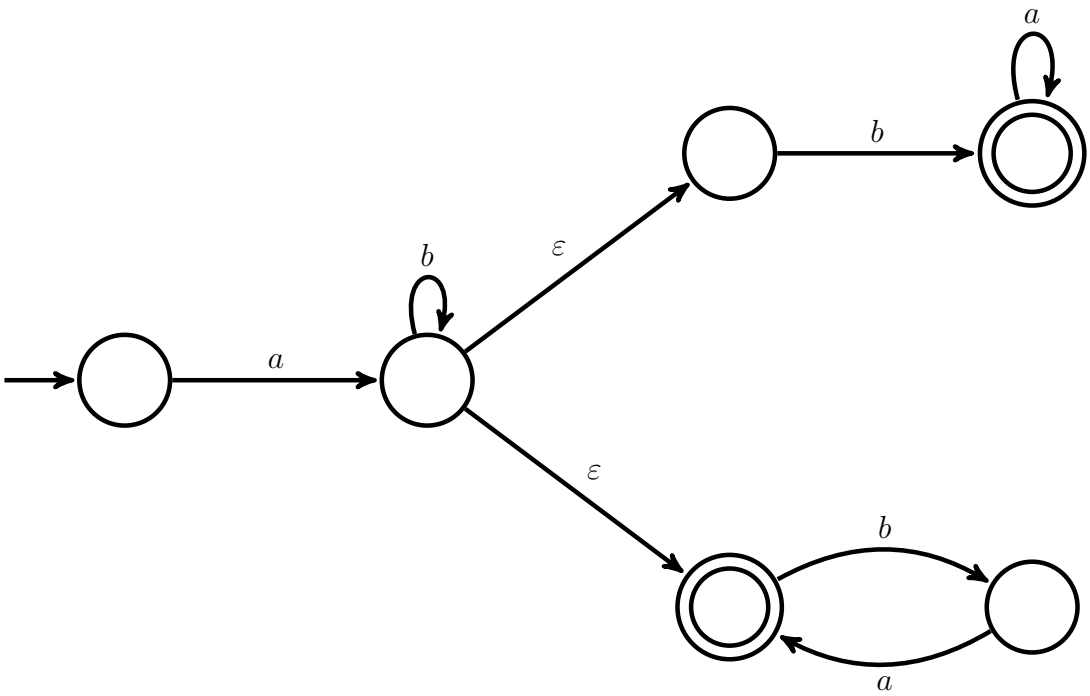
**Exercise 4-5:** For the regular expression $(aaa \cup aba)^* bab^*$, find an equivalent NFA.

**Solution:**




**Exercise 4-6:** Give an NFA defined by the regular expression $ab^*(ba^* \cup (ba)^*)$

**Solution:**



**Exercise 4-7:** Let $\Sigma = \{a, b\}$. A language is described by the regular expression: $(a \cup aba \cup ba)^*$

**a)** Give a string that is in this language.

**b)** Give a string that is NOT in this language.

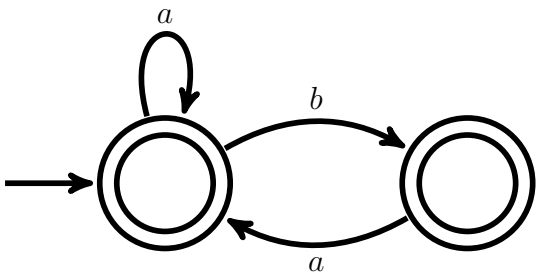
**c)** Give an NFA that recognizes this language.

**Solution:**

**a)** $ababa$

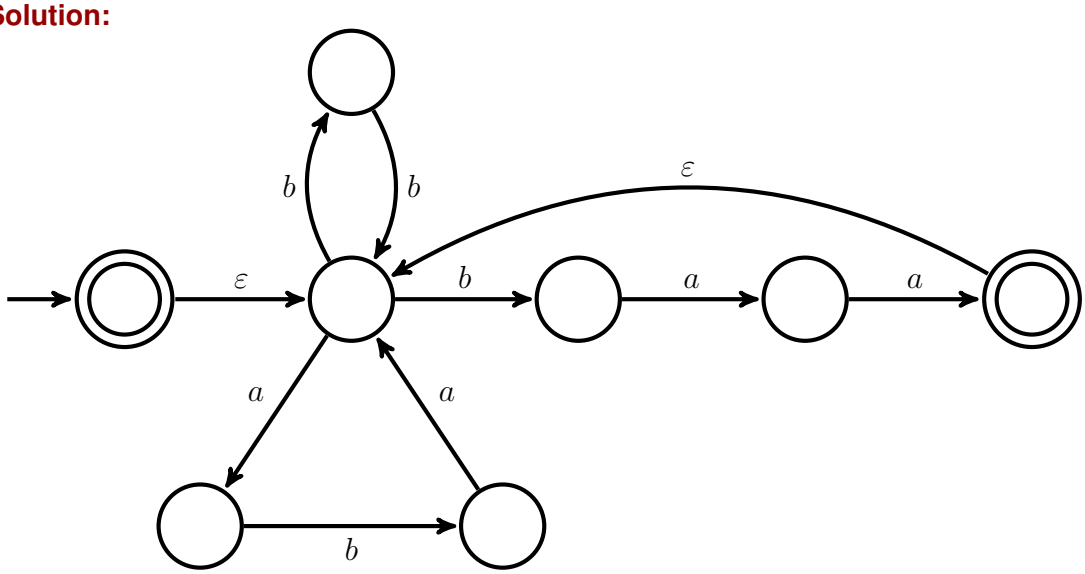
**b)** bb

**c)**



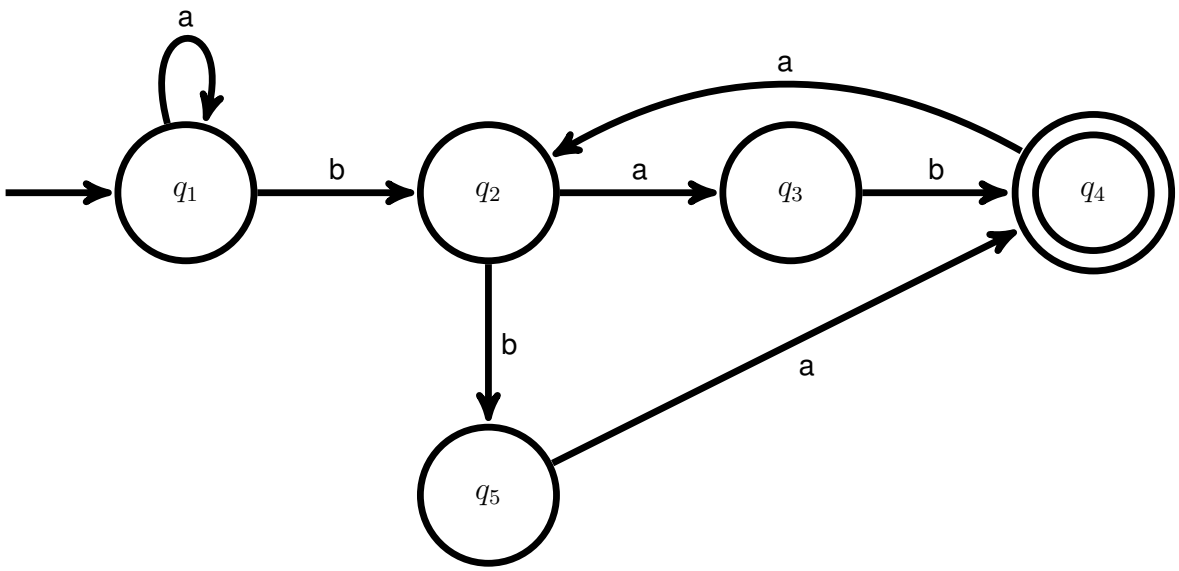

**Exercise 4-8:** For the regular expression $\left[(bb \cup aba)^* baa\right]^*$, find an equivalent NFA.

**Solution:**




**Exercise 4-9:**

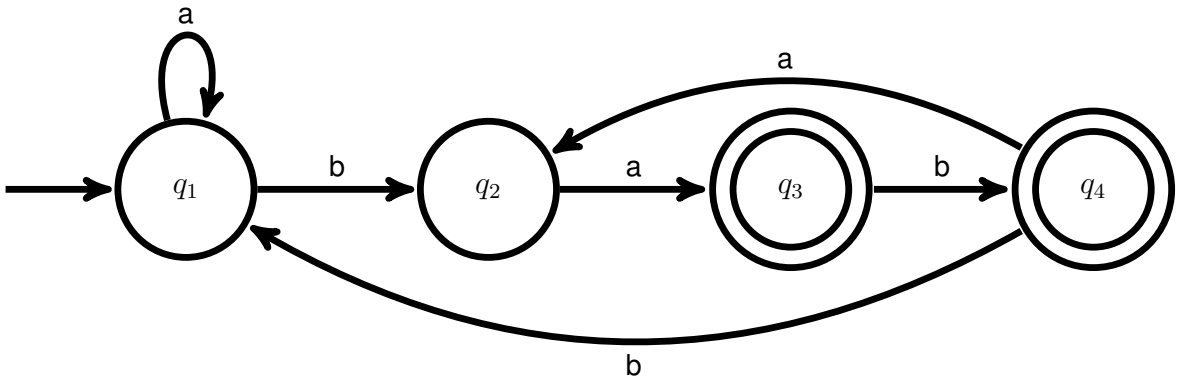
Find a regular expression equivalent to the language recognized by the following NFA:




**Solution:**

$$a^*b\big(aba \cup baa\big)^*\big(ab \cup ba\big)$$

**Exercise 4-10:** Find a regular expression equivalent to the language recognized by the following NFA:




**Solution:**

$$\big[a \cup b(aba)^*abb\big]^*b\big(aba\big)^*\big(a \cup ab\big)$$

# Week 5– Pumping Lemma

**Exercise 5-1:** Find an NFA that recognizes the language over $\Sigma = \{0, 1\}$:
$\{w \mid w$ has an equal number of occurrences of 01 and 10 as substrings$\}$

**Exercise 5-2:** Find an NFA that recognizes the language over $\Sigma = \{0, 1\}$:

**a)** $\{0^n 1^n | n \geqslant 0\}$

**b)** $\{w \mid w$ has an equal number of 0's and 1's$\}$

This looks similar to the previous question, but actually it is impossible. We need to store information to recognize these, but NFA's have no memory. So these two are **non-regular** languages.

**Theorem: (Pumping Lemma)** If $A$ is a regular language, then there is a number $p$ (the pumping length) where, if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, satisfying the following conditions:

- for each $i \geqslant 0, \quad xy^i z \in A$,
- $|y| > 0$
- $|xy| \leqslant p$

Note that $|s|$ is the length of string $s$, $y^i$ means that $i$ copies of $y$ are concatenated together, and $y^0 = \varepsilon$.

**Proof Idea:** Let $p$ be the number of states. Let $s = s_1 s_2 \cdots s_n$ be a string with $n \geqslant p$. The number of states the automaton will enter is $p + 1$ so at least one state is repeated by pigeonhole principle. Lets call the index of these states $f$ *(first)* and $\ell$ *(last)*. In that case

$$x = s_1 s_2 \cdots s_{f-1}, \quad y = s_f s_{f+1} \cdots s_{\ell-1}, \quad z = s_\ell s_{\ell+1} \cdots s_n$$

The string $y$ takes the DFA from the same state to the same state and the machine accepts $xyz$ therefore it must accept $xy^i z$.
Also, $f \neq \ell$ therefore $|y| > 0$ and $|xy| \leqslant p$.

**Exercise 5-3:** Show that $A = \{0^n 1^n | n \geqslant 0\}$ is nonregular.

**Exercise 5-4:** Show that $A = \{1^{n!} | n \geqslant 0\}$ is nonregular.

**Exercise 5-5:** Show that the language $A = \{w \mid w = w^R\}$ over $\Sigma = \{0, 1\}$ is non-regular.
(Here, $w^R$ shows the reverse of the string $w$, for example $10111^R = 11101$)

**Solution:** Suppose $A$ is regular. Let the pumping length be $p$. Choose $w = 0^p 10^p$.
We know that $w = xyz$ and $|xy| \leqslant p$ therefore $y$ consists of zeros only.
$y = 0^k$ where $k \leqslant p$.
$w = xyz \in A$ but $xyyz = 0^{p+k} 10^p \notin A$ for any possible choice of $y$. So there is a contradiction.
$A$ is non-regular by pumping lemma.

**Exercise 5-6:** Show that the language $A = \{0^n 1^{2n} 0^n\}$ is not regular.

**Solution:**
Assume $A$ is regular. Then, there is a pumping length $p$. Choose $s = 0^p 1^{2p} 0^p$. The length of $s$ is greater than $p$ so we should be able to pump it.

$$s = 0^p 1^{2p} 0^p = xyz$$

If $y$ consists of $1's$ only or $0's$ only, $xy^i z \notin A$. If $y$ contains both 0's and 1's, then the resulting string is not of the form $0^n 1^{2n} 0^n$ (they are out of order) so again $xy^i z \notin A$. Therefore $s$ cannot be pumped. So $A$ is not regular.

**Exercise 5-7:** Show that the language $A = \{a^{2n} b^{n+2} c^{n-2} \mid n \geqslant 2\}$ is not regular.

**Solution:** Assume $A$ is regular. Let pumping length be $p$.
Consider

$$s = a^{2p} b^{p+2} c^{p-2}$$

According to pumping lemma, we can find $x, y, z$ such that $s = xyz$. We also know that $|xy| \leqslant p$ therefore $y$ consists of $a's$ only. Suppose $y = a^k$. In this case

$$xy^i z = a^{2p+ik}\, b^{p+2}\, c^{p-2} \notin A$$

We have a contradiction, so $A$ is not regular.

# Week 6– Context Free Grammars

A grammar consists of a collection of substitution rules, also called productions. Each rule is of the form $A \to string$ where $A$ is a variable and the string is made of variables and terminals. The variable symbols often are represented by capital letters. The terminals are represented by lowercase letters, numbers, or special symbols. One variable is designated as the start variable. The sequence of substitutions is called derivation.
A language is context-free if there is a context-free grammar $G$ with $L = L(G)$.

**Example:** $S \to 0S1$ generates the language $0^n1^n$.
$S \to \varepsilon$
We can also express this as: $S \to 0S1 \mid \varepsilon$.

**Example:** $S \to aSb \mid SS \mid \varepsilon$. generates strings like $aabb$, $abab$, $aababb$ etc. We can think of those as open and close parenthesis.

**Example:** $S \to ABC$
$A \to a \mid the$
$B \to fish \mid bird$
$C \to flies \mid swims$

### Formal Definition of a Context-Free Grammar

A context-free grammar is 4-tuple $(V, \Sigma, R, S)$ where

- $V$ is a finite set called the variables,
- $\Sigma$ is a finite set, disjoint from $V$, called the terminals,
- $R$ is a finite set of rules, with each rule connecting a variable and a string of variables and terminals.
- $S \in V$ is the start variable.

**Exercise 6-1:** Design a CFG that generates palindromes of even length over $\Sigma = \{0, 1\}$.

(**Answer:** $S \to 0S0 \mid 1S1 \mid \varepsilon$)

**Exercise 6-2:** Design a CFG that generates palindromes over $\Sigma = \{0, 1\}$.

(**Answer:** $S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$)

### Ambiguity

Sometimes a grammar can generate the same string in several different ways. Such a string will have several different parse trees and thus several different meanings.
A derivation of a string $w$ in a grammar $G$ is a leftmost derivation if at every step the leftmost remaining variable is the one replaced.
A string $w$ is derived ambiguously in context-free grammar $G$ if it has two or more different leftmost derivations. Grammar $G$ is ambiguous if it generates some string ambiguously.

**Example:** $E \to E + E \mid E \times E \mid (E) \mid a$ generates $a + a \times a$ in two different ways. (They have different parse trees) So this grammar is ambiguous.

**Example:** $E \rightarrow E + T \mid T$
$T \rightarrow T \times F \mid F$
$F \rightarrow (E) \mid a$
generates the same language, but is not ambiguous.
Sometimes when we have an ambiguous grammar we can find an unambiguous grammar that generates the same language. Some context-free languages, however, can be generated only by ambiguous grammars. Such languages are called inherently ambiguous.

**Simplifying Grammars** We can simplify grammars by modifying and deleting some rules. For example:

- Eliminate $A \rightarrow \varepsilon$

$S \rightarrow 00A$
$A \rightarrow 1 \mid 0A1 \mid \varepsilon$
is equivalent to
$S \rightarrow 00A \mid 00$
$A \rightarrow 1 \mid 0A1 \mid 01$

- Eliminate $A \rightarrow B$

$S \rightarrow A1 \mid AB$
$A \rightarrow B \mid 11$
$B \rightarrow 000 \mid 101$
is equivalent to
$S \rightarrow A1 \mid AB \mid B1 \mid BB$
$A \rightarrow 11$
$B \rightarrow 000 \mid 101$

**Chomsky Normal Form**

A context-free grammar is in **Chomsky normal form** if every rule is of the form

$$A \rightarrow BC$$
$$A \rightarrow a$$

where $a$ is any terminal and $A, B$, and $C$ are any variables-except that $B$ and $C$ may not be the start variable. In addition we permit the rule $S \rightarrow \varepsilon$, where $S$ is the start variable.

**Theorem:** Any context-free language is generated by a context-free grammar in Chomsky normal form.
**Proof Idea:** We can convert any grammar $G$ into Chomsky normal form.
First, we add a new start variable. Then, we eliminate all $\varepsilon$ rules of the form $A \rightarrow \varepsilon$. We also eliminate all unit rules of the form $A \rightarrow B$. In both cases we patch up the grammar to be sure that it still generates the same language. Finally, we convert the remaining rules into the proper form.

**Exercise 6-3:** Eliminate rules containing $\varepsilon$ from the following grammar:

$$S \to AB \mid 00A$$
$$A \to BB \mid 0 \mid A11B \mid \varepsilon$$
$$B \to 1B \mid 1A1$$

**Solution:**

$$S \to AB \mid 00A \mid B \mid 00$$
$$A \to BB \mid 0 \mid A11B \mid 11B$$
$$B \to 1B \mid 1A1 \mid 11$$

**Exercise 6-4:** Give a CFG that generates the following language over $\Sigma = \{0, 1\}$:

$$\{w \mid w \text{ is of odd length and contains more 0's than 1's.}\}$$

**Solution:**

$$S \to T0T$$
$$T \to 0T1 \mid 1T0 \mid 0T0 \mid TT \mid \varepsilon$$

**Exercise 6-5:** Give a CFG generating the following language over $\Sigma = \{0, 1\}$:

$$\{w \mid w \text{ is of even length and starts and ends with the same symbol.}\}$$

**Solution:**

$$S \to 0A0 \mid 1A1 \mid \varepsilon$$
$$A \to 00A \mid 01A \mid 10A \mid 11A \mid \varepsilon$$

**Exercise 6-6:**
Give a CFG generating the following language over $\Sigma = \{0, 1\}$:

$$\{w \mid w \text{ is of odd length and contains at least two 0's.}\}$$

**Solution:**

$$S \to A0A0A \mid A0B0B \mid B0A0B \mid B0B0A$$
$$A \to 0B \mid 1B$$
$$B \to \varepsilon \mid 00B \mid 01B \mid 10B \mid 11B$$

(Here, $A$ is any string of odd length and $B$ is any string of even length.)

**Exercise 6-7:** Find a CFG that generates the language $A = \{a^n b^{n+2} \mid n \geqslant 0\}$

**Solution:**

$$S \to aSb \mid A$$
$$A \to bb$$

**Exercise 6-8:** Find a CFG that generates the language $B = \{a^n b^m \mid m > n\}$

**Solution:**

$$S \to aSb \mid A$$
$$A \to bA \mid b$$

**Exercise 6-9:** Convert the following CFG to Chomsky Normal Form:

$$S \to aSa \mid AA$$
$$A \to AB \mid B \mid \varepsilon$$
$$B \to a \mid b \mid \varepsilon$$

**Solution:**

$$S_0 \to CD \mid CC \mid AA \mid AB \mid BA \mid BB \mid a \mid b \mid \varepsilon$$
$$S \to CD \mid AA \mid AB \mid BA \mid BB \mid CC$$
$$C \to a$$
$$D \to SC \mid AC$$
$$A \to AB \mid BB$$
$$B \to a \mid b$$

**Exercise 6-10:** Consider the following CFG:

$$S \to A \mid AB \mid DA$$
$$A \to aAa \mid bB \mid \varepsilon$$
$$B \to D \mid b \mid CC$$
$$C \to c \mid AC \mid BC \mid \varepsilon$$
$$D \to Aa \mid Bb \mid CD$$

Give a derivation for $abbccabc$

**Solution:**

$$S \to AB \to aAaB \to aAaCC \to aAaCc \to aAaBCc \to aAabCc \to aAabc$$

$$abBabc \to abCCabc \to abBCCabc \to abbCCabc \to abbcCabc \to abbccabc$$

**Exercise 6-11:** Let $G$ :

$$S \to AB$$
$$A \to BB \mid B11B$$
$$B \to 0B0 \mid 1A1 \mid \varepsilon$$

**a)** Give an example of a string in $L(G)$.

**b)** Give an example of a string not in $L(G)$.

**c)** Convert it into Chomsky normal form.

**Solution:**

**a)** 00

**b)** 1

**c)**

$$S \to AB \mid CB \mid DB \mid BB \mid BC \mid BD \mid YE \mid ZF \mid YY \mid ZZ \mid \varepsilon$$
$$A \to BB \mid BC \mid BD$$
$$B \to YE \mid ZF \mid YY \mid ZZ$$
$$C \to DB$$
$$D \to ZZ$$
$$E \to BY$$
$$F \to AZ \mid CZ \mid DZ \mid BZ$$
$$Y \to 0$$
$$Z \to 1$$

**Exercise 6-12:** Find a context free grammar that recognizes the language over $\Sigma = \{0, 1\}$ consisting of strings of odd length where first, middle and last symbols are the same.

**Solution:**

$S \to 0A0 \mid 1B1 \mid 0 \mid 1$
$A \to 0A0 \mid 1A1 \mid 0A1 \mid 1A0 \mid 0$
$B \to 0B0 \mid 1B1 \mid 0B1 \mid 1B0 \mid 1$

OR

$S \to 0A0 \mid 1B1 \mid 0 \mid 1$
$A \to CAC \mid 0$
$B \to CBC \mid 1$
$C \to 0 \mid 1$

**Exercise 6-13:** Convert the following grammar into Chomsky normal form:
$S \to AbA$
$A \to aB \mid bBb \mid \varepsilon$
$B \to A \mid ba$

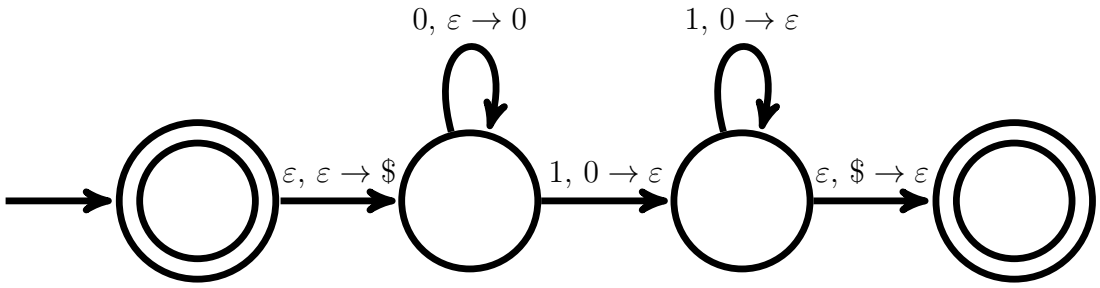
**Solution:** First eliminate $B \to A$, then eliminate $A \to \varepsilon$ and then break triples to obtain:

$S \to YA \mid AY \mid AC \mid b$
$A \to XB \mid YD \mid XA \mid YE \mid YY \mid a$
$B \to YX$
$C \to YA$
$D \to BY$
$E \to AY$
$X \to a$
$Y \to b$

# Week 7– Pushdown Automata

Now we will add memory to an NFA and obtain a more powerful machine that can recognize some extra languages.
A pushdown automaton (PDA) can write symbols on the stack and read them back later. Stack is a LIFO (Last In, First Out) memory. Writing a symbol is called *pushing*.
We can only read the symbol on the top of the stack. Once we read it, it is removed. This is called *popping*.
A PDA chooses the next stage based on the current state, the input *and* the stack.
Using stack, we can design an automaton that recognizes the language $0^n1^n$. Using a special symbol \$, we can test whether the stack is empty or not.

**Exercise 7-1:** The PDA that recognizes $\{0^n1^n \mid n \geqslant 0\}$ is:




**Formal definition of a Pushdown Automaton**

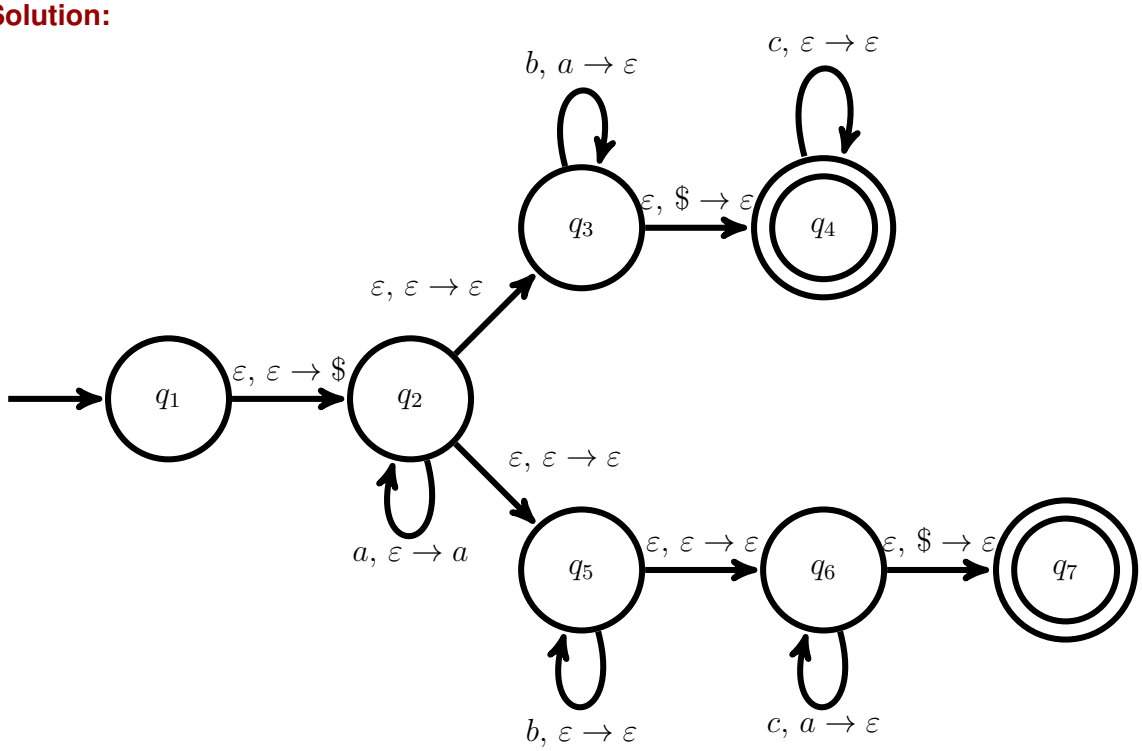
A **pushdown automaton** is a 6 tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where $Q, \Sigma, \Gamma$ and $F$ are all finite sets, and

- $Q$ is the set of states,
- $\Sigma$ is the input alphabet,
- $\Gamma$ is the stack alphabet,
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept states.

**Exercise 7-2:** Find a PDA that recognizes the language:

$$\{a^i b^j c^k \mid i, j, k \geqslant 0 \text{ and } \big(i = j \text{ or } i = k\big)\}$$

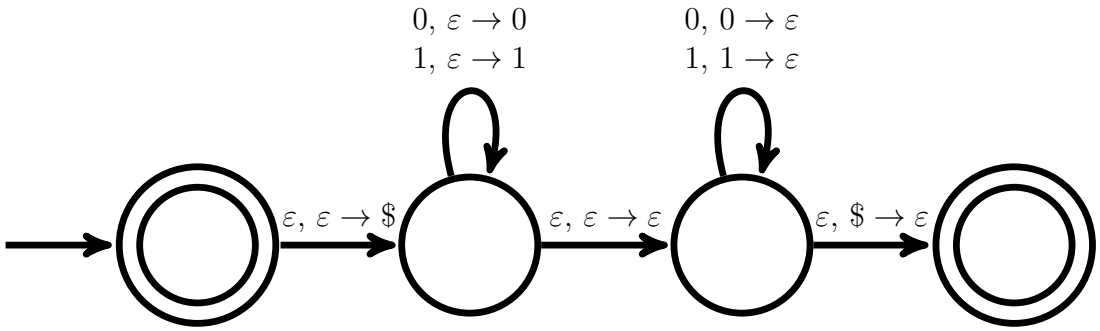
**Solution:**




**Exercise 7-3:** Find a CFG that generates the same language.

**Exercise 7-4:** Find a PDA that recognizes the language $\{ww^R \mid w \in \{0,1\}^*\}$ where $w^R$ indicates the symmetric of the string $w$ from right to left.

**Solution:**



**Theorem:** A language is context free if and only if some pushdown automaton recognizes it.

**Lemma:** If a pushdown automaton recognizes some language, then it is context free.

**Theorem:** If $A_{pq}$ generates $x$, then $x$ can bring $P$ from $p$ with empty stack to $q$ with empty stack.

**Theorem:** If $x$ can bring $P$ from $p$ with empty stack to $q$ with empty stack, $A_{pq}$ generates $x$.

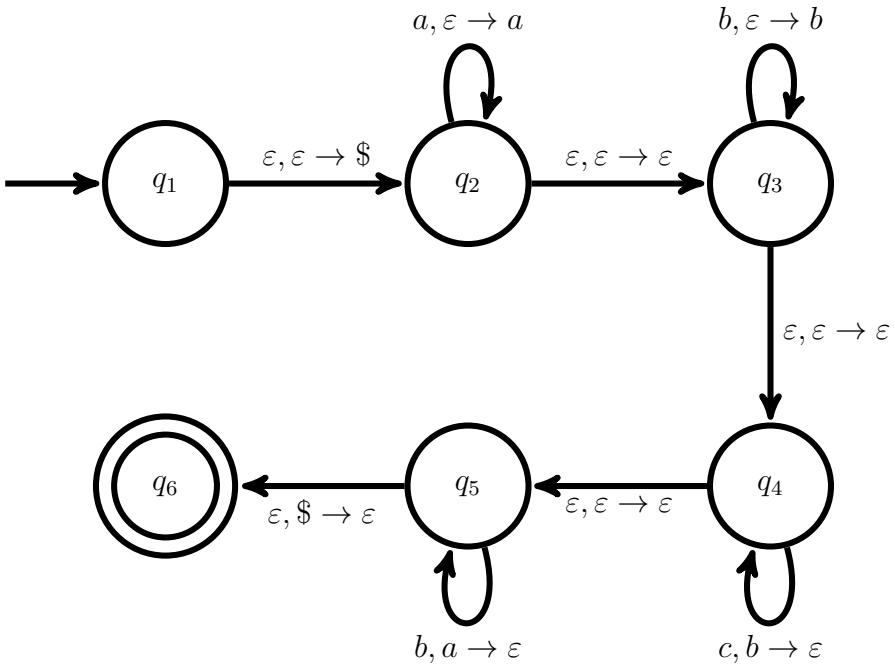
**Corollary:** Every regular language is context free.

**Exercise 7-5:** Give *either* a PDA (pushdown automaton) *or* a CFG (context free grammar) for the language $\{a^n b^m c^m b^n \mid n, m \geqslant 0\}$ over $\Sigma = \{a, b, c\}$

**Solution:**

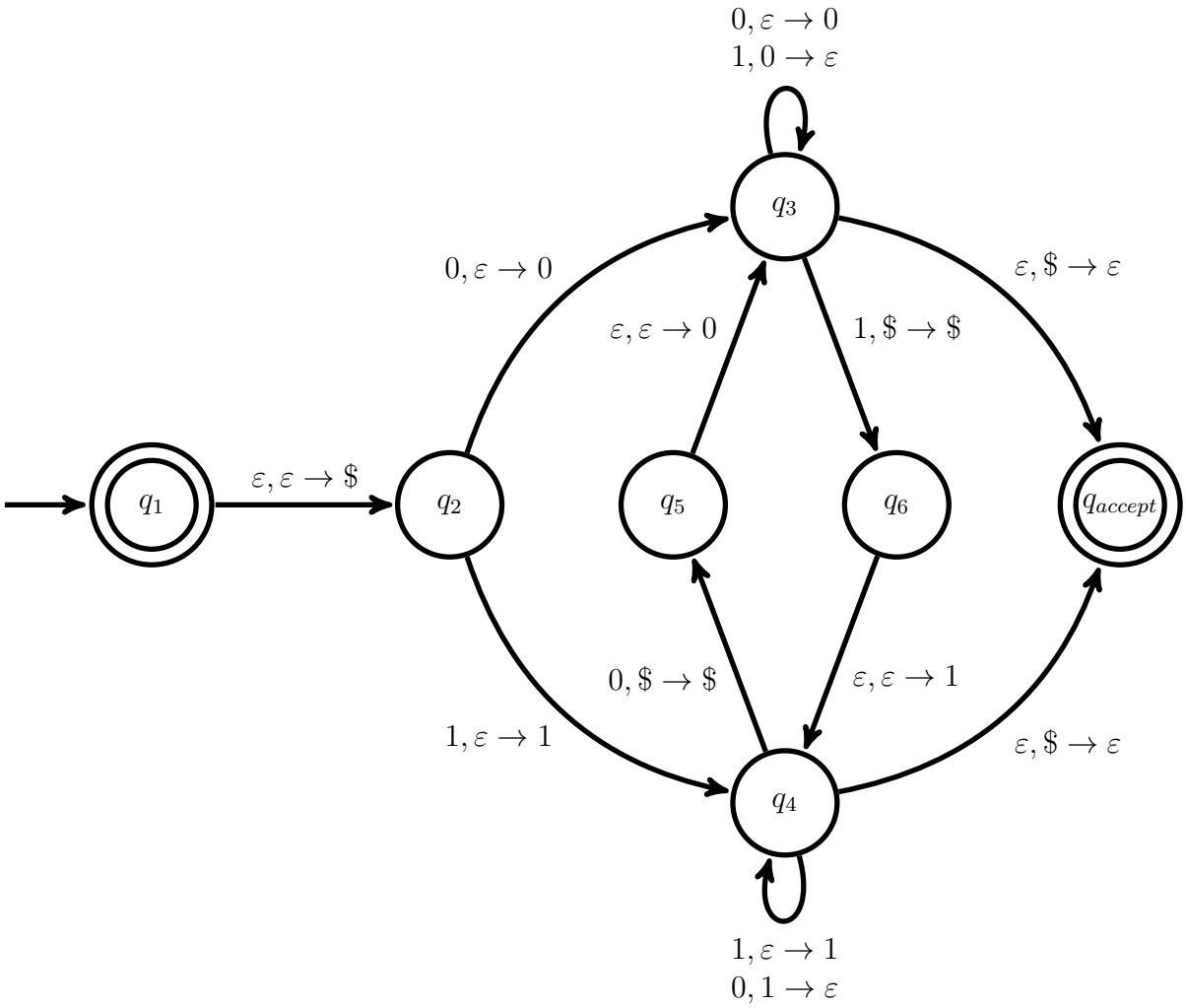
$$S \to aSb \mid T$$
$$T \to bTc \mid \varepsilon$$

OR



**Exercise 7-6:** Give a PDA (pushdown automaton) for the language over $\Sigma = \{0, 1\}$ made of strings that contain equal numbers of 0's and 1's.

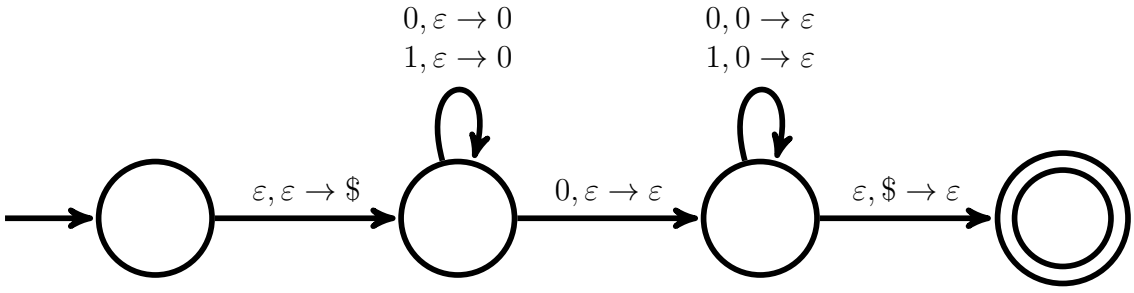
**Solution:**




State $q_3$: Up to now, more $0$'s than $1$'s arrived. Stack contains $0$'s.
State $q_4$: Up to now, more $1$'s than $0$'s arrived. Stack contains $1$'s.

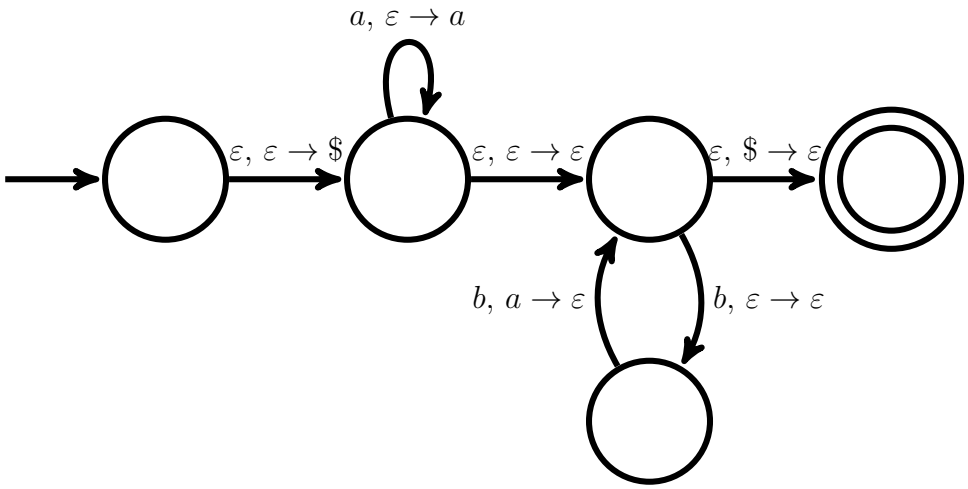
**Exercise 7-7:** Find a pushdown automaton (PDA) for the language over $\Sigma = \{0, 1\}$:
$\{w \mid$ length of $w$ is odd and its middle symbol is 0$\}$

**Solution:**



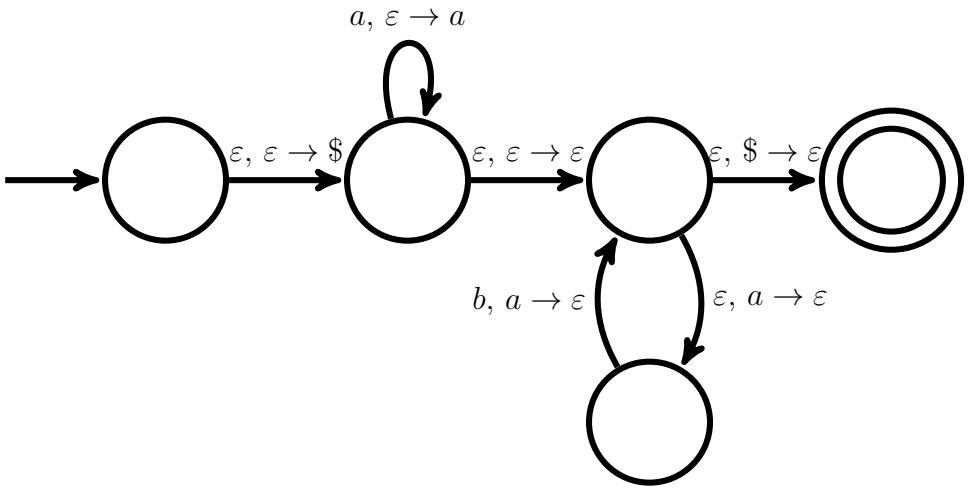
**Exercise 7-8:**
Give a PDA (pushdown automaton) that recognizes the language $\{a^n b^{2n} \mid n \geqslant 0\}$ over $\Sigma = \{a, b\}$.

**Solution:**



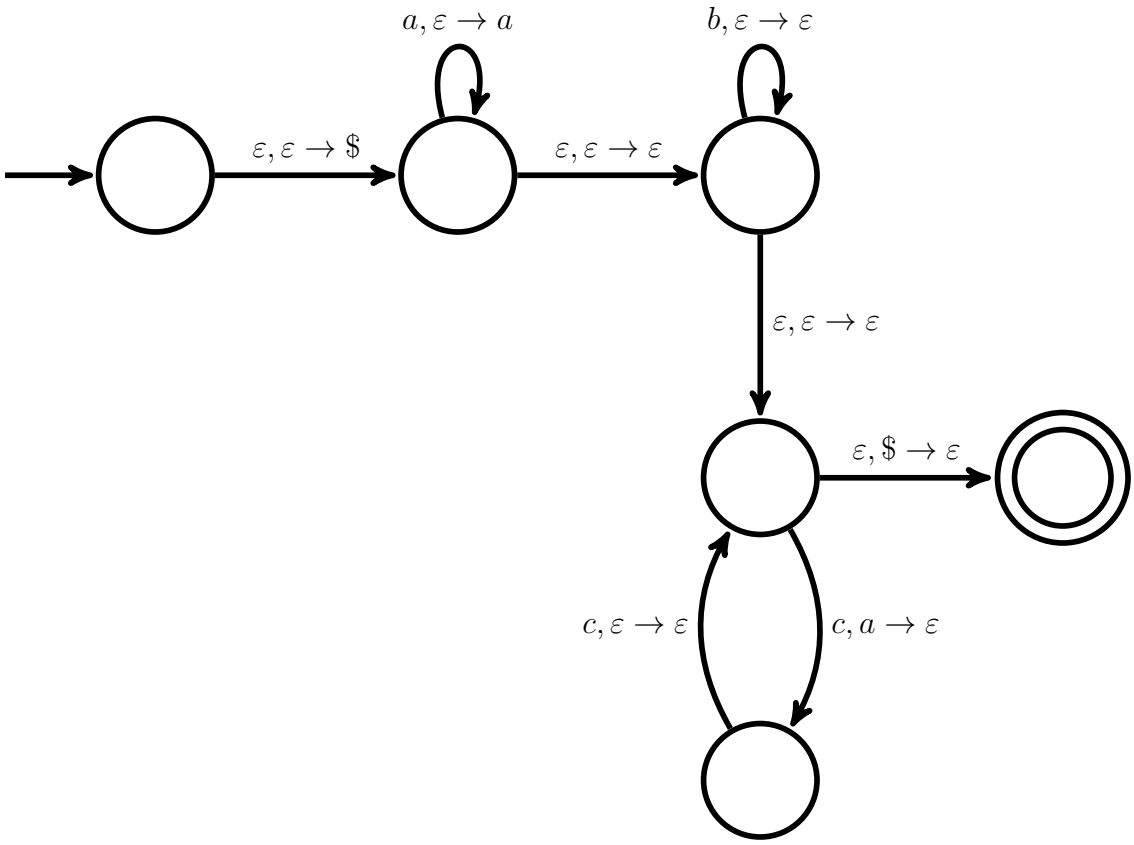

**Exercise 7-9:** Give a PDA (pushdown automaton) that recognizes the language $\{a^{2n} b^n \mid n \geqslant 0\}$ over $\Sigma = \{a, b\}$.

**Solution:**

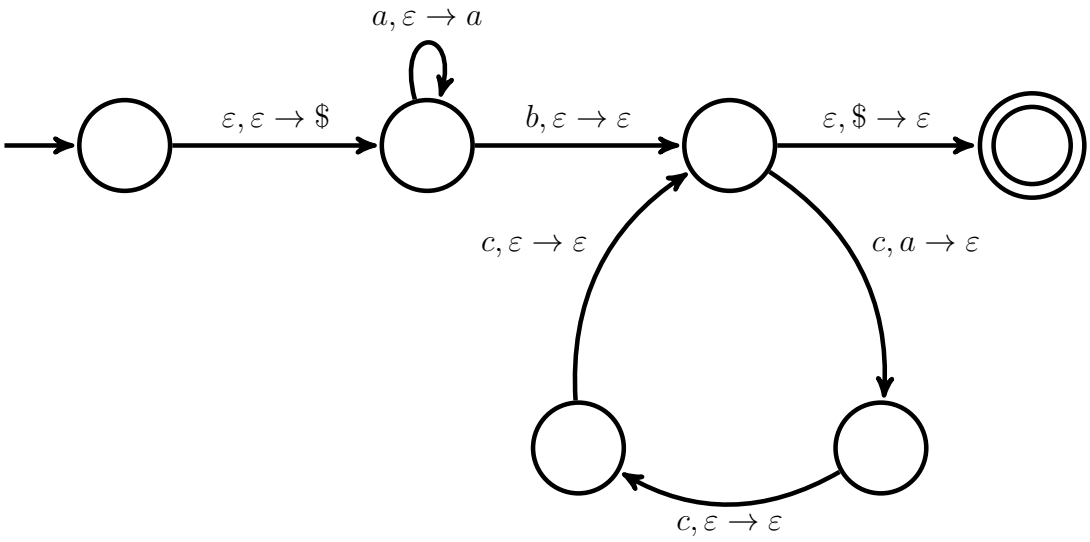


**Exercise 7-10:** Give a PDA that recognizes the language $A = \{a^n b^m c^{2n} \mid m, n \geqslant 0\}$

**Solution:**



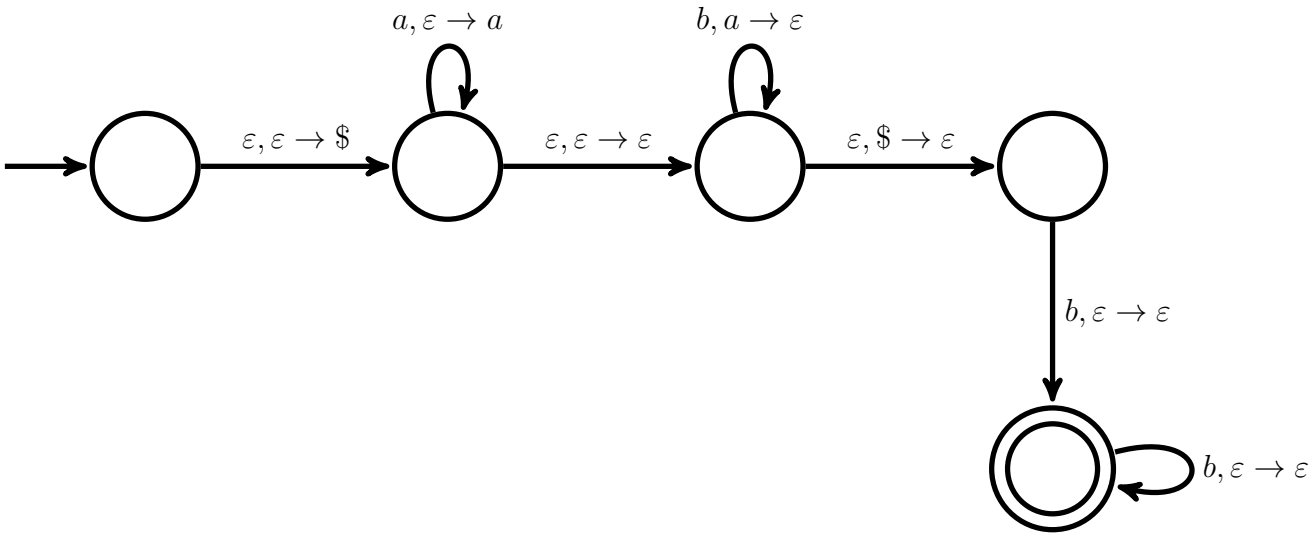

**Exercise 7-11:** What language does the following Pushdown Automaton (PDA) recognize?




**Solution:**
$a^n b c^{3n}, n \geqslant 0$

**Exercise 7-12:** Find a PDA that recognizes the language $\{a^n b^m \mid m > n\}$.

**Solution:**

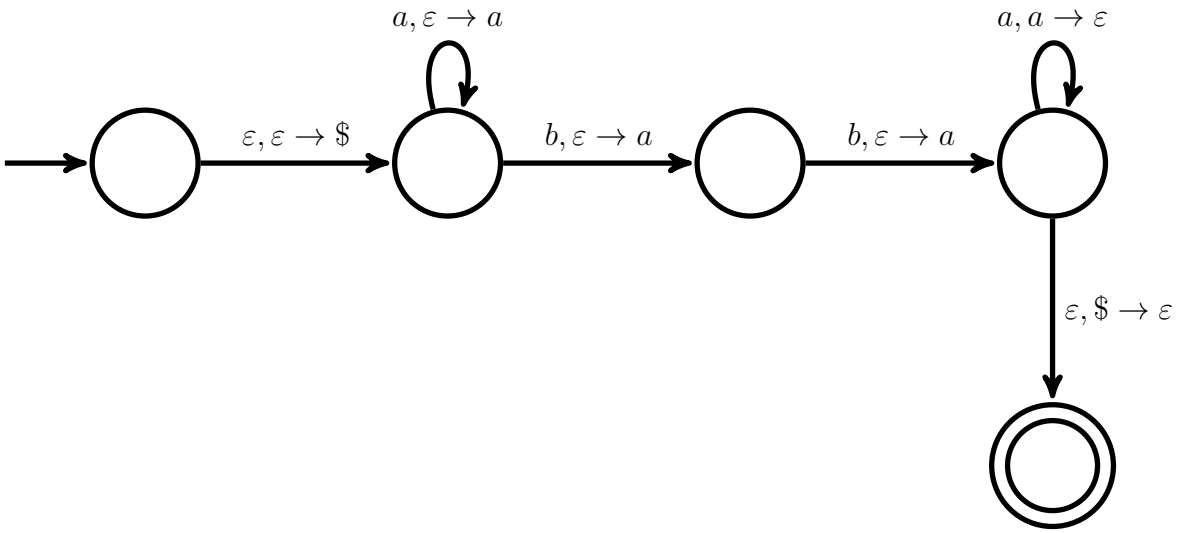



**Exercise 7-13:**

**a)** Find a PDA that recognizes the language $\{a^n b^2 a^{n+2} \mid n \geqslant 0\}$

**b)** Find a CFG generating the same language.

**Solution:**



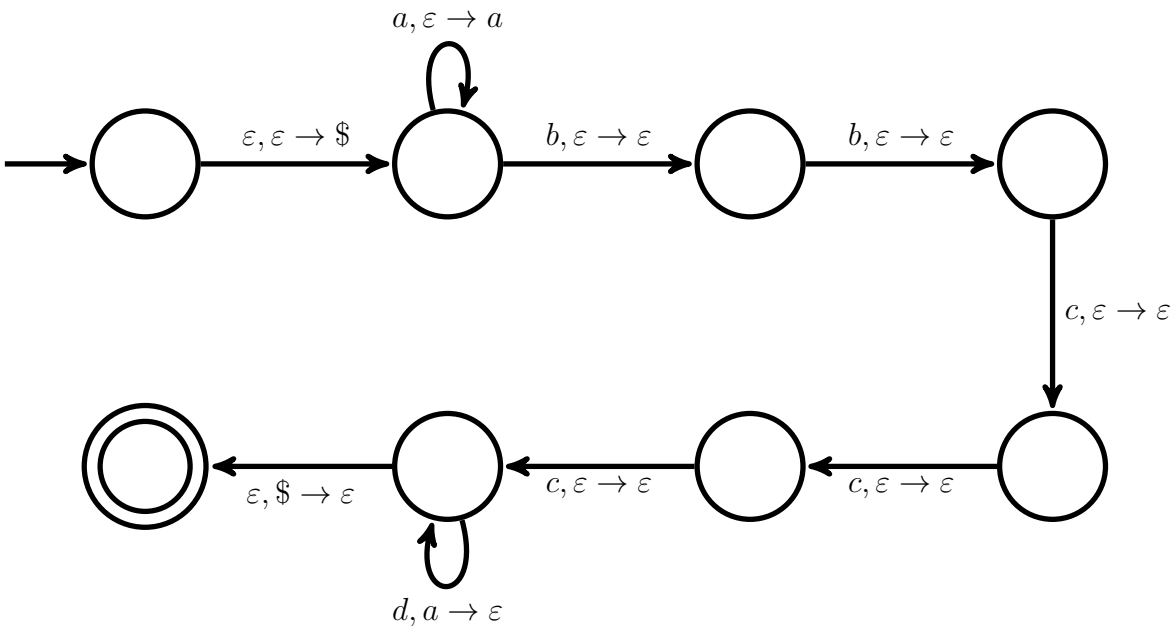

$S \to aSa \mid b^2 a^2$

**Exercise 7-14:**

**a)** Find a PDA that recognizes the language $\{a^n b^2 c^3 d^n \mid n \geqslant 0\}$

**b)** Find a CFG generating the same language.

**Solution:**



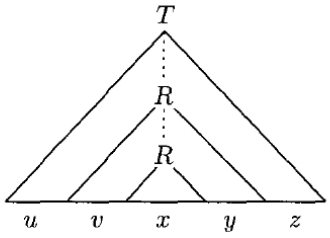

$S \to aSd \mid b^2c^3$

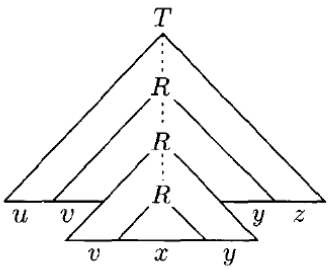
# Week 8– Non-Context-Free Languages

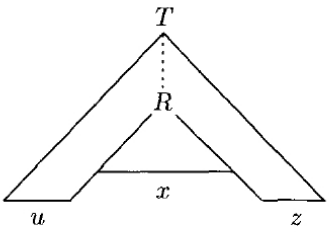
**Theorem:** (Pumping lemma for context-free languages) If $A$ is a context-free language, then there is a number $p$ (the pumping length) where, if $s$ is any string in A of length at least $p$, then $s$ may be divided into five pieces $s = uvxyz$ satisfying the conditions:

- for each $i \geqslant 0$, $uv^ixy^iz \in A$,
- $|vy| > 0$, and
- $|vxy| \leqslant p$.

When $s$ is being divided into $uvxyz$, condition 2 says that either $v$ or $y$ is not the empty string. Otherwise the theorem would be trivially true.
Condition 3 states that the pieces $v, x$, and $y$ have length at most $p$.
**Proof Idea:** Let $A$ be a CFL and let $G$ be a CFG that generates it.
Let $s$ be a very long string in $A$. Its parse tree is so tall that some variable symbol $R$ must repeat because of the pigeonhole principle.
If the tree is binary, the relationship between the string length and height is: $|s| = 2^h$.
Then we can replace the subtree under the second $R$ with the subtree under the first $R$ and still get a legal parse tree.




Therefore, we may cut $s$ into five pieces $uvxyz$ as the figure indicates, and we may repeat the second and fourth pieces and obtain a string still in the language.
In other words:

$$uv^ixy^iz \in A \text{ for any } i \geqslant 0$$

**Example:** $T \rightarrow 11R11$
$R \rightarrow 00R00 \mid 111$

**Exercise 8-1:** Use the pumping lemma to show that the language $\{a^n b^n c^n \mid n \geqslant 0\}$ is not context free.

(**Answer:** $s = a^p b^p c^p$)

$v$ and $y$ contain either single type of symbol, or several types of symbols. Both are impossible.

**Exercise 8-2:** Let $A = \{ww \mid w \in \{0,1\}^*\}$. Use pumping lemma to show that $A$ is not CFL.

(**Answer:** $s = 0^p 1 0^p 1$ does not work, but $s = 0^p 1^p 0^p 1^p$ works)

**Exercise 8-3:** Let $B = \{a^n\, b\, a^{2n}\, b\, a^{3n} \mid n \geqslant 0\}$. Use pumping lemma to show that $B$ is not CFL.

(**Answer:** $s = a^p b a^{2p} b a^{3p}$, v and y can't contain b)

**Exercise 8-4:** Let $A = \{a^n\, b^m\, a^{3n} \mid n, m \geqslant 0\}$. Is A context-free? What does the pumping lemma say?

(**Answer:** $Yes!$)

**Exercise 8-5:** Let $A = \{a^n\, b\, a^m\, b\, a^{n+m} \mid n, m \geqslant 0\}$. Is A context-free? What does the pumping lemma say?

(**Answer:** $Yes!$)

**Exercise 8-6:** Let $A = \{a^n b^{n+1} c^{4n} \mid n \geqslant 0\}$. Show that $A$ is not a context-free language.

**Solution:** Suppose $A$ is context free. Let $p$ be the pumping length. Choose $w$ as $w = a^p b^{p+1} c^{4p}$. How can we choose $v$ and $y$ such that $a^p b^{p+1} c^{4p} = uvxyz$?

**1)** They contain more than one symbol.

In this case, the pumped string $uv^2xy^2z$ will have symbols out of order. For example, it will contain $a$'s after $b$'s. Therefore $uv^2xy^2z \notin A$.

**2)** They contain a single symbol.

In that case, we can pump at most two of the symbols $\{a, b, c\}$. The remaining one will have the same power in the pumped string. For example, if we choose $v = a$ and $y = b$ we obtain

$uv^2xy^2z = a^{n+1}b^{n+2}c^{4n} \notin A$
Therefore we cannot pump this string. By pumping lemma, $A$ is not context free.

**Exercise 8-7:** Let $A = \{a^{n!} \mid n \geqslant 0\}$. Use pumping lemma to show that $A$ is not a context-free language.

**Solution:** Let the pumping length be $p$. Choose the string as $s = a^{p!}$. Now divide into five parts as $s = uvxyz$. Any choice for $v$ and $y$ consist of $a$'s only. We know that

$$|vxy| \leqslant p$$

therefore

$$|uv^2xy^2z| \leqslant p! + p$$

Clearly,

$$p! + p < (p+1)!$$

therefore

$$uv^2xy^2z \neq a^{(p+1)!} \quad \Rightarrow \quad uv^2xy^2z \notin A$$

So we cannot pump this string.

**Exercise 8-8:** Let $B = \{a^{n^2} \mid n \geqslant 0\}$. Use pumping lemma to show that $B$ is not a context-free language.

**Solution:** Let the pumping length be $p$. Choose the string as $s = a^{p^2}$. Now divide into five parts as $s = uvxyz$. Any choice for $v$ and $y$ consist of $a$'s only. We know that

$$|vxy| \leqslant p$$

therefore

$$|uv^2xy^2z| \leqslant p^2 + p$$

Clearly,

$$p^2 + p < (p+1)^2$$

therefore

$$uv^2xy^2z \neq a^{(p+1)^2} \quad \Rightarrow \quad uv^2xy^2z \notin A$$

So we cannot pump this string.

**Exercise 8-9:** Let $A = \{a^n b^n c^{2n} d^{2n} \mid n \geqslant 0\}$. Show that $A$ is not a context-free language.

**Solution:**
Let pumping length be $p$. Let $s = a^p b^p c^{2p} d^{2p}$. Clearly, length of $s$ is greater than $p$. By pumping lemma,

$$s = uvxyz$$

Case 1: $v$ and $y$ contain one type of symbol only.
Then, $uv^2xy^2z \notin A$ because powers are different.

Case 2: $v$ or $y$ contain more than one type of symbol.
Then, $uv^2xy^2z \notin A$ may contain equal numbers of $a, b, c, d$ but they will be out of order, so $uv^2xy^2z \notin A$.

$s$ cannot be pumped, therefore $A$ is not context free.

**Exercise 8-10:** Consider the following languages over $\Sigma = \{a, b, c, d\}$:

- $\{a^n b^m c^m d^n\}$
- $\{a^n b^n c^m d^m\}$
- $\{a^n b^m c^n d^m\}$

**a)** Which one is non-context free?

**b)** Show that it is not using pumping lemma.

**Solution:** **a)** $\{a^n b^m c^n d^m\}$

**b)** Suppose the language is context free. Let $p$ be the pumping length. Choose $s$ as $s = a^n b^m c^n d^n$ where $m, n > p$. According to pumping lemma, we can find $v, y$ such that

$$a^n b^m c^n d^n = uvxyz$$

If $v$ or $y$ contain more than one type of symbol, clearly $uv^2xy^2z \notin A$ because symbols are out of order. Therefore $v$ and $y$ can consist of a single symbol only.
In this case, we have two choices. $v$ must consist of $a's$ and $y$ must consist of $c's$, or $v$ must consist of $b's$ and $y$ must consist of $d's$. Both cases violate the rule $|vxy| < p$
Therefore we cannot pump this string so the language is not context free.

**Exercise 8-11:**
Use the pumping lemma to show that the following language is not context free:

$$A = \{a^n bca^{4n} ba^n \mid n \geqslant 0\}$$

**Solution:** Suppose A is context free. Let p be the pumping length. Choose $s$ as $s = a^p bca^{4p} ba^p$. According to pumping lemma, we can find$v, y$ such that

$$a^p bca^{4p} ba^p = uvxyz$$

If $v$ or $y$ contain $b$ or $c$, clearly $uv^2xy^2z \notin A$ because it is not in the given format. Therefore $v$ and $y$ can consist of $a's$ only.
But in this case, we can pump at most two of the parts including $a$. The size of the third part will remain the same, so pumped string will not be in $A$. For example, $v = a, y = aaaa$ will result in

$$uv^2xy^2z = a^{p+1} bca^{4p+4} ba^p \notin A$$

Therefore we cannot pump this string and $A$ is not context free.

**Exercise 8-12:**
Use the pumping lemma to show that the following language is not context free:

$$A = \{a^n b^m c^{2n} d^{2m} \mid m, n \geqslant 0\}$$

**Solution:** Suppose A is context free. Let p be the pumping length. Choose $s$ as $s = a^n b^m c^{2n} d^{2m}$ and $n, m \geqslant p$. According to pumping lemma, we can find $v, y$ such that

$$a^n b^m c^{2n} d^{2m} = uvxyz$$

If $v$ or $y$ contain more than one type of symbol, clearly $uv^2xy^2z \notin A$ because it is not in the given format. Therefore $v$ and $y$ can consist of a single symbol only.
If $v$ consists of $a's$, then $y$ must consist of (twice as many) $c's$. If $v$ consists of $b's$, then $y$ must consist of (twice as many) $d's$.
In bothe cases, $|vxy| > p$ and the third condition of pumping lemma is violated.
Therefore we cannot pump this string and $A$ is not context free.

# Week 9– Turing Machines

The Turing machine is like a finite automaton but it has unlimited memory, and it can access any part of the memory. Thus, it is equivalent to a computer, but still, there are problems a Turing machine cannot solve.

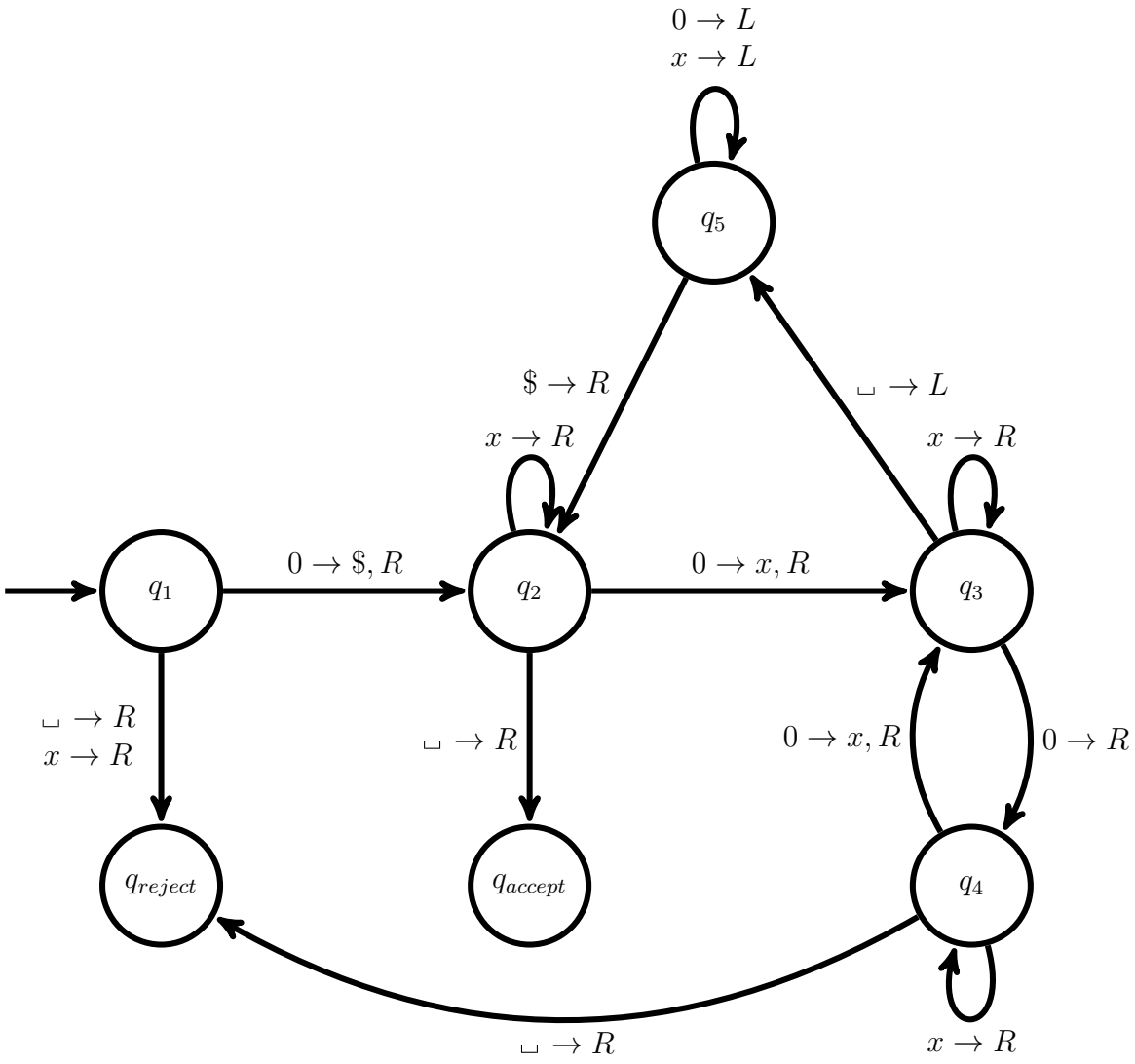
**Formal Definition of a Turing Machine**

A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where $Q, \Sigma, \Gamma$ are all finite sets and:

1. $Q$ is the set of states,
2. $\Sigma$ is the input alphabet not containing the blank symbol $\sqcup$ ,
3. $\Gamma$ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

A Turing machine $M$ receives its input on the leftmost $n$ squares of the tape, and the rest of the tape is blank. The head cannot move to the left of start square.
The computation proceeds according to the transition function. The head can move left or right, it can read and write on the tape. If the machine enters an accept or reject state, the computation halts. If neither occurs, it continues forever.
Machines that never loop are called deciders because they always make a decision to accept or reject. They halt on all inputs.
We call a language Turing-recognizable if some Turing machine recognizes it, and we call a language Turing-decidable or simply decidable if some Turing machine decides it.

**Exercise 9-1:** Describe a Turing machine that decides $A = \{0^{2^n} \mid n \geqslant 0\}$, the language consisting of all strings of $0$'s whose length is a power of 2.
(Here, $ is used to determine the beginning of the tape)




An alternative to schematic representation is a pseudo-code algorithm:

1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, accept.
3. If in stage 1 the tape contained more than a single 0 and the number of 0's was odd, reject.
4. Return the head to the left-hand end of the tape.
5. Go to stage 1.

**Exercise 9-2:** Describe a Turing Machine deciding the language $C = \{a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geqslant 1\}$.

**Solution:**

1. Scan the input from left to right to determine whether $i, j, k \geqslant 1$ and reject otherwise.
2. Return the head to the left-hand end of the tape.
3. Cross off an $a$ and scan to the right until $b$ occurs. Shuttle between the $b$'s and the $c$'s, crossing off one of each until all $b$'s are gone. If all $c$'s have been crossed off and some $b$'s remain, reject.
4. Restore the crossed off $b$'s and repeat stage 3 if there is another $a$ to cross off. If all $a$'s have been crossed off, determine whether all $c$'s also have been crossed off. If yes, accept, otherwise, reject.

**Exercise 9-3:** Here, a TM solves what is called the element distinctness problem. It is given a list of strings over $\{0, 1\}$ separated by #s and its job is to accept if all the strings are different. The language is;

$$\{\#w_1\#w_2\#\cdots\#w_n \mid \text{ each } w_i \in \{0,1\}^* \text{ and } w_i \neq w_j \text{ for each } i \neq j\}$$

**Solution:**

1. If the first symbol is blank, accept. If it is 0 or 1, reject. If it is a #, put a mark on it and continue to stage 2.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, there was only one string so accept.
3. By zig-zagging, compare the two strings to the right of the $\dot{\#}$ ( marked #s). If they are equal, reject.
4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so accept.
5. Go to Stage 3.

**Exercise 9-4:** Describe a Turing machine that recognizes the language $A = \{a^n b^{n+1} c^{4n} \mid n > 0\}$

**Solution:**

1. Sweep from left to right. IF any symbol is out of order (for example, $a$'s after $b$'s) REJECT.
2. Go to start. Search for $a$.
   IF found, cross it. (Replace by $\times$)
   ELSE, Go to 6.
3. Search for $b$.
   IF found, cross it.
   ELSE, REJECT.
4. Repeat 4 times:
   Search for $c$.
   IF found, cross it.
   ELSE, REJECT.
5. Go to 2.
6. Sweep from left to right.
   IF there is one and only one $b$ AND there is no $c$, ACCEPT.
   ELSE, REJECT.

**Exercise 9-5:** Define the language $L$ as: $\langle A, k \rangle$ where $A$ is an NFA, $k$ is an integer and $A$ rejects all strings of length $\leqslant k$.
Show that $L$ is decidable. (Hint: Give a TM for $L$ that halts)

**Solution:**

1. Simulate $A$ on the TM.
2. Mark start state.
3. For $i = 1$ to $k$
   Mark all states that can be reached from marked states in one step.
   If an accept state is marked, Return REJECT.
   EndFor
4. Return ACCEPT.

Another idea (that is bad style but still works) is to make a list of all strings of length $\leqslant k$, give them one by one to our simulation of $A$ and reject if it accepts any one of them.

**Exercise 9-6:** Give description of a Turing machine $M$ that, given an input from $\{0, 1, \#\}^*$, eliminates all $\#'s$. For example, given input $011\#01\#0$, the tape should contain $011010$ when $M$ halts.

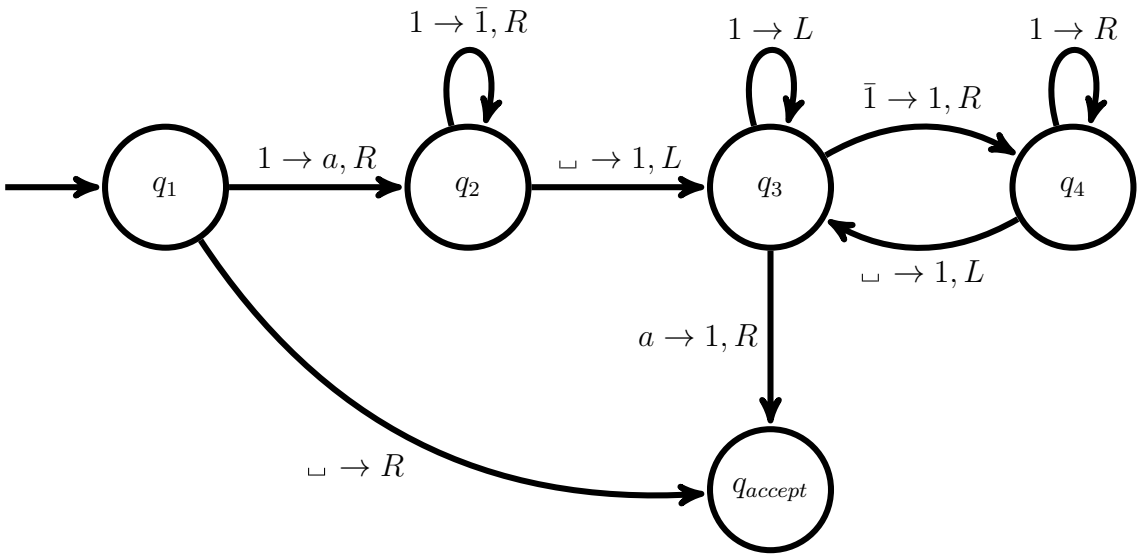
**Solution:**

1. Move right until meeting the first #. If there is no #, accept.
2. Move right, read symbol, move left, write symbol, move right. (This moves one symbol one unit left, the symbol could be $0, 1$, # or blank).
3. If symbol is blank, go to 4.
   Else, go to 2. (This moves all symbols to the right of # one unit left).
4. Move the head to the start and go to 1.

**Exercise 9-7:** Give description of a Turing machine $M$ that, given an input from $\{1\}^*$, doubles the number of symbols. For example, given input $111$, the tape should contain $111111$ when $M$ halts.

**Solution:**

1. If the first symbol is blank, accept. If it is 1, put a mark on it and move right. Put a mark on all 1's until meeting blank.
2. Move left until meeting the first $\bar{1}$ (marked 1). Erase the mark. If there's no marked 1, accept.
3. Move right until meeting blank. Replace blank with 1.
4. Go to 2.




(Here, $a$ denotes the very first $\bar{1}$)

**Exercise 9-8:** Describe a Turing Machine deciding the language $A = \{0^n 1^{n^2} \mid n \geqslant 1\}$.

**Solution:**

1. Sweep from left to right. IF there is no 0, or no 1, or if they are out of order, REJECT.

2. Move head to start. Search for 0.
   IF found,
   Mark it.
   Move right until blank. Write #.
   Go to 2.
   *//Write as many #'s as there are 0's.*

3. Search for #.
   IF found
   Mark it.
   Unmark all marked 0's.
   Shuttle between 0's and 1's. Mark one 1 for each one 0.
   IF 0 is not found, Go to 3.
   IF 1 is not found, REJECT.

4. ELSE (# not found)
   Move head to start. Search for 1.
   IF found, REJECT.
   ELSE, ACCEPT.

**Exercise 9-9:** Let A be the language in $\{0, 1\}^*$ made of strings where the number of zeros is at least 3 times the number of ones. Describe a Turing Machine recognizing A.

**Solution:**

1. Move head to start. Search for 1.
   IF found, cross it. (Replace by $\times$)
   ELSE, ACCEPT.

2. Repeat 3 times:
   Move head to start. Search for 0.
   IF found, cross it. (Replace by $\times$)
   ELSE, REJECT.

3. Go to 1.

**Exercise 9-10:**
Describe a Turing Machine that gives the output $x_m, x_{m-1}, \ldots, x_2, x_1$ for the input $x_1, x_2, \ldots, x_{m-1}, x_m$.

**Solution:**

1. Move to Start. Find the first element that is not crossed. Call it $a$. Cross it.
   If not found, Go To 4.
2. Move right until meeting blank or cross. Move left. Read the element. Call it $b$.
3. Swap $a$ and $b$ if they are different. Cross $a$ and $b$. Go To 1.
4. Sweep from left to right. Restore all crossed elements. Return.

**Exercise 9-11:**
Describe a Turing Machine that gives the output $x_1, x_2, \ldots, x_m, x_1, x_2, \ldots, x_m$ for the input $x_1, x_2, \ldots, x_m$.

**Solution:**

1. Move to Start. Find the first element that is not crossed. Cross it.
   If not found, Go To 4.
2. Move right until meeting blank.
3. Write the element found in 1. Cross it. Go To 1.
4. Sweep from left to right. Restore all crossed elements. Return.

# Week 10– Variants of Turing Machines

### Multitape Turing Machines

A multitape Turing machine has several tapes. Each tape has its own head for reading and writing. Initially the input appears on tape 1, and the others start out blank.
Multitape Turing machines appear to be more powerful than ordinary Turing machines, but actually they are equivalent they recognize the same languages.

**Theorem:** Every multitape Turing machine has an equivalent single-tape Turing machine.

**Proof:** Say that $M$ has $k$ tapes. Then $S$ simulates the effect of $k$ tapes by storing their information on its single tape. It uses the new symbol # as a delimiter to separate the contents of the different tapes. In addition to the contents of these tapes, $S$ must keep track of the locations of the heads. It does so by writing a tape symbol with a dot above it to mark the place where the head on that tape would be. Think of these as ”virtual” tapes and heads.

### Nondeterministic Turing Machines

The computation of a nondeterministic Turing machine is a tree whose branches correspond to different possibilities for the machine. If some branch of the computation leads to the accept state, the machine accepts its input.

**Theorem:** Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

**Proof Idea:** We can simulate any nondeterministic TM $N$ with a deterministic TM $D$. The idea behind the simulation is to have $D$ try all possible branches of $N$'s nondeterministic computation. If $D$ ever finds the accept state on one of these branches, $D$ accepts.
Here, we have to be careful and use breadth-first search, not depth-first search.

### Enumerators

Loosely defined, an enumerator is a Turing machine with an attached printer. The Turing machine can use that printer as an output device to print strings. Every time the Turing machine wants to add a string to the list, it sends the string to the printer.
An enumerator E starts with a blank input tape. If the enumerator doesn't halt, it may print an infinite list of strings. The language enumerated by E is the collection of all the strings that it eventually prints out. Moreover, E may generate the strings of the language in any order, possibly with repetitions. Now we are ready to develop the connection between enumerators and Turing recognizable languages.

**Theorem:** A language is Turing-recognizable if and only if some enumerator enumerates it.

**Proof:** First we show that if we have an enumerator $E$ that enumerates a language A, a TM $M$ recognizes A. The TM $M$ works in the following way.

1. Run $E$. Every time that $E$ outputs a string, compare it with $w$.
2. If $w$ ever appears in the output of $E$, accept.

Clearly, $M$ accepts those strings that appear on $E$'s list.

Now we do the other direction. If TM $M$ recognizes a language A, we can construct the following enumerator $E$ for $A$. Say that $s_1, s_2, \ldots$ is a list of all possible strings in $\Sigma^*$.
Ignore the input.

1. Repeat the following for $i = 1, 2, 3, \ldots$
2. Run $M$ for $i$ steps on each input, $s_1, s_2, \ldots, s_i$
3. If any computations accept, print out the corresponding $s_j$."

If $M$ accepts a particular string $s$, eventually it will appear on the list generated by $E$. (It may appear on the list infinitely many times, because it runs $s_1$ one step, then $s_1, s_2$ two steps, then $s_1, s_2, s_3$ three steps etc.)

**Equivalence With Other Models**

Any two computational models that satisfy certain reasonable requirements can simulate one another and hence are equivalent in power. This equivalence phenomenon has an important philosophical corollary.
Computational models may have a certain arbitrariness but the underlying class of algorithms is natural.

**Exercise 10-1:** A Turing machine with doubly infinite tape has a single tape, but its tape is infinite to the left and to the right. The tape is initially blank except for the input. Show that this is equivalent to an ordinary Turing machine.

**Solution:** We can divide the double tape into two parts: positive and negative. Then, we can map it into normal Turing tape by zig-zagging and placing positive squares to odd squares and negative squares to even squares as follows:

$$f(n) = \begin{cases} 2n-1 & n > 0 \\ -2n & n < 0 \end{cases}$$

Double Tape

| $\cdots$ | $-3$ | $-2$ | $-1$ | $1$ | $2$ | $3$ | $\cdots$ |
|---|---|---|---|---|---|---|---|

Normal Tape

| $1$ | $-1$ | $2$ | $-2$ | $3$ | $-3$ | $\cdots$ |
|---|---|---|---|---|---|---|

OR

We can use the multi-tape idea for double tape and separate the contents by a $\#$:

Double Tape

| $\cdots$ | $-3$ | $-2$ | $-1$ | $1$ | $2$ | $3$ | $\cdots$ |
|---|---|---|---|---|---|---|---|

Normal Tape

| $1$ | $2$ | $\cdots$ | $n$ | $\#$ | $-1$ | $-2$ | $\cdots$ |
|---|---|---|---|---|---|---|---|

Note that in this case the # moves as the program proceeds.

**Exercise 10-2:** Let $E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \varnothing\}$.

In other words, $A$ is a DFA that does not accept any string.
Show that $E_{DFA}$ is a decidable language.

**Solution:** The following Turing machine decides this language. (Try to build a path from accept state backwards to start state)

On input $A$ where $A$ is a DFA:

1. Mark the accept states of $A$. If there is no accept state, ACCEPT.
2. Repeat until no new states get marked:
3. If a state has an arrow pointing to a marked state, mark it.
4. If start state is marked, REJECT; otherwise, ACCEPT.

**Exercise 10-3:**
Let $E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \varnothing\}$.

In other words, $G$ is a CFG that does not generate any string.
Show that $E_{CFG}$ is a decidable language.

**Solution:** The following Turing machine decides this language. (Try to build a path from terminals backwards to $S$)

On input $G$, where $G$ is a CFG:

1. Use Chomsky form. Mark all terminal symbols. If there is no terminal, ACCEPT.
2. Repeat until no new variables get marked:
3. Mark any variable $A$ if there is a rule $A \to a$, OR if there is a rule $A \to BC$ and both $B$ and $C$ are marked.
4. If $S$ is not marked, ACCEPT; otherwise, REJECT.

**Exercise 10-4:** Describe a Turing machine that recognizes the language over $\Sigma = \{a, b, c\}$
$A = \{w \mid$ the number of $a$'s is twice the number of $c$'s $\}$.

**Solution:**

1. If the tape is empty, ACCEPT.
2. Sweep from left to right. Find the first $a$. Mark it. If there is no $a$, Go To 4.
   Sweep from left to right. Find the first $a$. Mark it. If there is no $a$, REJECT.
3. Sweep from left to right. Find the first $c$. Mark it. Go To 2. If there is no $c$, REJECT.
4. Sweep from left to right. If there is no $c$, ACCEPT. If there is a $c$, REJECT.

**Exercise 10-5:** Describe a Turing Machine that calculates $n^2$. It uses an input alphabet of single letter, for example $a$. The input is $n$ symbols and the output is $n^2$ symbols on the tape.

**Solution:**

1. Place $b$ at the end of input.
2. Repeat the following:
3. Sweep from left until $b$. Mark the first $a$.
   If there is no unmarked $a$, erase $b$, RETURN.
4. Repeat the following:
5. Go left. Sweep until $b$. If you see $a$, make it $c$. If you see marked $a$,
   make it marked $c$. In both cases put an $a$ to the end of the tape.
6. Transform all $c$'s back to $a$'s.

**Exercise 10-6:** If there is a state in a DFA that is not reachable from the start state, it is redundant. Consider the set of all DFA's that do not contain any redundant state. Think of it as a language. Is this language decidable?

**Solution:**
The following TM decides it:

1. Mark start state. Repeat:
2. Mark all states reachable from marked states.
3. If no new states are marked, search for an unmarked state.
4. If there is an unmarked state, REJECT. Otherwise, ACCEPT

**Exercise 10-7:** Let $A$ be the language consisting of Turing Machines which halt in 100 steps or less for some input.
Is $A$ decidable?
Hint: Reading the input also requires steps.

**Solution:**
The following TM decides it:

1. Check all inputs with 100 symbols or less.
2. If it halts for any one of these ACCEPT.
3. Otherwise REJECT.

**Exercise 10-8:** What language does the following Turing machine recognize? Is it a decider? (Input alphabet is: $\Sigma = \{a, b\}$)

1. Sweep from left to right. IF there is any $a$ after the first $b$, REJECT.
2. Move head to start. Search for $a$.
   IF found, cross it. (Replace by $\times$)
   ELSE, Go to 6.
3. Search for $b$.
   IF found, cross it.
   ELSE, REJECT.
4. Search for $b$.
   IF found, cross it.
   ELSE, REJECT.
5. Go to 2.
6. Move head to start. Search for $b$.
   IF found, ACCEPT.
   ELSE, REJECT.

**Solution:** It is a decider, it can not enter any infinite loops.
Its language is: $\{a^n b^m \mid m \geqslant 2n + 1\}$.

**Exercise 10-9:**
Let $A$ be the language in $\{0,1\}^*$ made of strings containing an equal number of 0's and 1's. Describe a Turing Machine recognizing $A$. Is it a decider?

**Solution:**

1. Move head to start. Search for 0.

2. IF there's a 0,
   Cross it.
   Move head to start.
   Search for 1.
   IF there is a 1,
   Cross it.
   Go to 1.
   ELSE
   REJECT.

3. IF there's no 0,
   Move head to start.
   Search for 1.
   IF there is a 1,
   REJECT.
   ELSE
   ACCEPT.

This machine stops after finitely many steps, because loops are repeated (at most) as many times as the number of symbols in the input. It is a decider.

**Exercise 10-10:**
Let $B$ be the language in $\{0,1\}^*$ made of strings containing more 0's than 1's. Describe a Turing Machine recognizing $B$. Is it a decider?

**Solution:**

1. Move head to start. Search for 0.

2. IF there's a 0,
   Cross it.
   Move head to start.
   Search for 1.
   IF there is a 1,
   Cross it.
   Go to 1.
   ELSE
   ACCEPT.

3. IF there's no 0,
   REJECT.

This machine stops after finitely many steps, because loops are repeated (at most) as many times as the number of symbols in the input. It is a decider.

# Week 11– Definition of Algorithm

Informally speaking, an algorithm is a collection of simple instructions for carrying out some task. Even though algorithms have had a long history in mathematics, the notion of algorithm itself was not defined precisely until the twentieth century.

**Hilbert's Problems**

In 1900, mathematician David Hilbert delivered a now-famous address at the International Congress of Mathematicians in Paris. In his lecture, he identified 23 mathematical problems and posed them as a challenge for the coming century. Hilbert's tenth problem was to devise an algorithm that tests whether a polynomial has an integral root.

**Question:** Let $p$ be a polynomial in many variables with integer coefficients. For example, $p = 3xy - 5y^2 + 6zx$. Does $p$ have an integer root?
This problem is Turing-recognizable but not decidable.
Consider the simpler case of a polynomial of one variable. Then, the following TM recognizes it: The input is a polynomial $p$ over the variable $x$.

1. Evaluate $p$ with $x$ set successively to the values $0, 1, -1, 2, -2, 3, 3, \ldots$
2. If at any point the polynomial evaluates to 0, accept.

This is a recognizer, but not a decider. (Why?)
We can convert it into a decider by restricting the range it searches to $|x| < \dfrac{k\, c_{max}}{c_1}$ where $k$ is the number of terms in the polynomial, $c_{max}$ is the coefficient with largest absolute value, and $c_1$ is the coefficient of the highest order term. If a root is not found within these bounds, the machine rejects. Matijasevic's theorem shows that calculating such bounds for multivariable polynomials is impossible.

**Algorithms and Turing Machines:** From now on, we will be interested in algorithms. So our descriptions of Turing Machines will be high-level.

**Exercise 11-1:** Let A be the language consisting of all strings representing undirected graphs that are connected. Recall that a graph is connected if every node can be reached from every other node by traveling along the edges of the graph. We write

$$A = \{G \mid G \text{ is a connected undirected graph}\}$$

The following is a high-level description of a TM that decides A. $M$ = On input ($G$), the encoding of a graph $G$:

1. Select the first node of $G$ and mark it.
2. Repeat the following stage until no new nodes are marked:
3. For each node in $G$, mark it if it is attached by an edge to a node that is already marked.
4. Scan all the nodes of $G$ to determine whether they all are marked. If they are, accept; otherwise, reject."

**Exercise 11-2:** Let $A =$ the set of all integer coefficient polynomials.
$B =$ the set of all functions $f : \mathbb{N} \to \mathbb{N}$.
$C =$ the set of all infinite strings on $\Sigma = \{0, 1, 2\}$.
Choose one of these sets. Claim that it is countable or uncountable. Prove your claim.

**Solution:** **a)** The set $\{\ldots, -1, 0, 1, 2, \ldots\}$ is countable. Therefore there are countably many $a_0$ terms in the polynomial. Similarly, there are countably many $a_1 x$ terms. In that way, we have to find countable union of countable sets which is countable. We can count them in the same way we counted rational numbers, by counting an infinite matrix.

**b)** We can think of each function as an infinite string. The set of such strings is uncountable. If we assume there is a list, we can generate an element that is not on the list using Cantor's diagonal idea. This contradicts the assumption that the list contains everything. This is the same idea we used in proving uncountability of real numbers on $[0, 1]$.

**c)** The same as **b)**.

**Exercise 11-3:** You are given a set of $n$ distinct positive integers. You want to determine if there are two integers $p, q$ in the set such that $p = q^2$.
Write an algorithm in pseudo-code for this problem. Show that it is in $P$.

**Solution:**
INPUT Integer $A[1], A[2], \ldots, A[n]$
For $i = 1$ to $n$
    For $j = 1$ to $n$
        If $A[i] * A[i] == A[j]$
            Return TRUE
        EndIf
    EndFor
EndFor
Return FALSE

This algorithm clearly does $\Theta(n^2)$ operations, so it is in P.

**Exercise 11-4:**
Let $A$ be the set of all $2 \times 2$ matrices with entries from $\mathbb{Q}$. Show that $A$ is countable.

**Solution:**
We know that $\mathbb{Q}$ is countable. Suppose a list of $\mathbb{Q}$ is $\{q_1, q_2, \ldots\}$. We can count $\mathbb{Q} \times \mathbb{Q}$ by making an infinite table and counting diagonally.
Suppose we obtain a list $S = \{s_1, s_2, \ldots\}$. Now make a table for $S \times S$. The list will give elements of $A = \mathbb{Q} \times \mathbb{Q} \times \mathbb{Q} \times \mathbb{Q}$.

**Exercise 11-5:** Let $B$ be the set of all finite strings based on the alphabet $\{0, 1, \ldots, 9\}$. Show that $B$ is countable.

**Solution:**
The following gives a list of all finite strings that can be written with this alphabet:

| *String* |
|---:|
| 0 |
| 1 |
| ⋮ |
| 9 |
| 00 |
| 01 |
| ⋮ |
| 99 |
| 000 |
| ⋮ |

# Week 12– Decidability and Halting Problem

### The acceptance problem for DFAs

We can represent a DFA by a string. (How?)
Then, the problem of whether a given DFA accepts a given string:

$$A_{DFA} = \{\langle B, w\rangle \mid \text{B is a DFA that accepts input string } w\}$$

can be expressed as a language. Is there a TM that decides this language?

YES.
On input $\langle B, w\rangle$, where $B$ is a DFA and $w$ is a string:

1. Simulate $B$ on input $w$.
2. If the simulation ends in an accept state, accept. If it ends in a nonaccepting state, reject.

**Exercise 12-1:** Define

$$A_{NFA} = \{\langle B, w\rangle \mid B \text{ is an NFA that accepts input string } w\}$$

Is $A_{NFA}$ decidable?

YES.
On input $\langle B, w\rangle$ where $B$ is an NFA, and $w$ is a string:

1. Convert NFA $B$ to an equivalent DFA $C$.
2. Run TM of previous exercise on input $\langle C, w\rangle$.
3. If M accepts, accept; otherwise, reject.

Similarly, we can determine whether a regular expression generates a given string by converting it into an NFA.

**Exercise 12-2:** Define

$$A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$$

Is $A_{CFG}$ decidable?

YES
Generating everything and comparing with $w$ is not a good idea, this process may or may not halt. We have to restrict the length of strings we check.

On input $\langle G, w \rangle$, where $G$ is a CFG and $w$ is a string:

1. Convert $G$ to an equivalent grammar in Chomsky normal form.
2. List all derivations with $2n - 1$ steps, where $n$ is the length of $w$, except if $n = 0$, then instead list all derivations with 1 step.
3. If any of these derivations generate $w$, accept; if not, reject.

### The Halting Problem

We can construct TM's that decides simpler languages. Now consider a larger class. Define:

$$A_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$$

Is this decidable?

NO!

**Theorem:** $A_{TM}$ is undecidable.
The following Turing machine recognizes $A_{TM}$

1. Simulate $M$ on input $w$.
2. If $M$ ever enters its accept state, ACCEPT. If $M$ ever enters its reject state, REJECT.

But it is not a decider, it may enter an infinite loop.
This is a Universal Turing Machine because it can simulate any other Turing machine from the description of that machine.

### Diagonalization

We have two infinite sets. Which one is larger? Are they always the same size?
If we have a one-to-one onto function between two sets (a 1-1 correspondence) they have the same size. By this definition, $\mathbb{N}$ and $2\mathbb{N}$ have the same size.
A set is countable if either it is finite or it has the same size as $\mathbb{N}$.

**Exercise 12-3:** Is $\mathbb{Z}$ countable?

**Exercise 12-4:** Is a union of countable sets countable?

**Exercise 12-5:** Is $\mathbb{Q}$ countable?
Make an infinite matrix. Count using the diagonals.

**Exercise 12-6:** Is $\mathbb{R}$ countable?
Suppose numbers on $[0, 1]$ are countable. Express them in binary notation. Make a list. Now, produce an element that is not in the list by choosing opposite of diagonal digits
$\Rightarrow$ NO.

**Remark:** There are uncountably many languages yet only countably many Turing machines. Each Turing machine can recognize a single language therefore some languages are not recognized by any Turing machine.
Now we are ready to prove the undecidability of $A_{TM}$.
Suppose that $H$ is a decider for $A_{TM}$.

$$H(\langle M, w\rangle) = \begin{cases} \text{ACCEPT} & \text{if } M \text{ accepts } w \\ REJECT & \text{if } M \text{ does not accept } w \end{cases}$$

Now we construct a new Turing machine $D$ with $H$ as a subroutine. This new TM calls $H$ to determine what $M$ does when the input to $M$ is its own description $\langle M\rangle$. Once $D$ has determined this information, it does the opposite.

$$D(\langle M\rangle) = \begin{cases} \text{ACCEPT} & \text{if } M \text{ does not accept } \langle M\rangle \\ REJECT & \text{if } M \text{ accepts } \langle M\rangle \end{cases}$$

What happens when we run $D$ with its own description $\langle D\rangle$ as input? In that case we get a contradiction. Thus neither $D$ nor $H$ can exist.

**A Turing-Unrecognizable Language**

**Theorem:** A language is decidable iff it is Turing-recognizable and co-Turing-recognizable. In other words, a language is decidable exactly when both it and its complement are Turing-recognizable.

**Proof Idea:** Suppose both $A$ and $\bar{A}$ are Turing Recognizable. Any given string $w$ is either in $A$ or in $\bar{A}$. Therefore if we run two machines in parallel, one of them will halt.

**Corollary:** $\overline{A_{TM}}$ is not Turing-recognizable.

**Exercise 12-7:** In Subset Sum problem, we are given a set of integers. The question is whether there is a subset of integers whose sum is 0. For example, for the set $A = \{-9, -5, -1, 3, 6, 18, 22\}$ the answer is YES, but for $B = \{-9, -5, -1, 3, 8, 18, 22\}$ the answer is NO.

**a)** Is Subset Sum in NP?

**b)** Is Subset Sum in P?

**Solution:**

**a)** YES. Given a solution, we can verify it in polynomial $(O(n))$ time.

**b)** NO. There is no known polynomial algorithm, we have to check all possible subsets and there are $2^n$ of them.

**Exercise 12-8:** You are given a set of $n$ positive integers. You have to find two integers $p, q$ from this set such that $p + q$ is as close as possible to 256.

**a)** Write an algorithm in pseudo code.

**b)** Find the time complexity of this algorithm.

**Solution:**

1. $p = n_1, q = n_2$
2. For $i = 1$ to $n - 1$
3. For $j = i + 1$ to $n$
4. If $|n_i + n_j - 256| < |p + q - 256|$
   $p = n_i, q = n_j$
5. EndFor
6. EndFor
7. Return $p, q$

This is clearly $O(n^2)$

**Exercise 12-9:** Consider the language consisting of all undirected graphs containing a triangle. Show that this language is in $P$.

**Solution:**

For any set of 3 nodes, check whether they are connected. We need $\binom{n}{3} = \frac{(n-1)(n-2)(n-3)}{6}$ operations. This is $O(n^3)$ so it is polynomial time.

**Exercise 12-10:**

STRING MATCHING:

You are given a string of length $n$. You are also given a word (a second string) of length $k$, where $k \leqslant n$. You want to determine if the word occurs in the string.

Show that the string matching problem is in $P$.

**Solution:**

```
INPUT String A[n], Word B[k]
For i = 1 to n − k + 1
        test = TRUE
        For j = 1 to k
                If A[i + j − 1] ≠ B[j]
                        test = FALSE
                        Break
                EndIf
        EndFor
        If test == TRUE
                Return TRUE
EndFor
Return FALSE
```

This algorithm uses $nk$ steps therefore it is $\Theta(n^2)$. So it is in $P$.

**Exercise 12-11:**

You are given $n$ distinct positive integers where $n \geqslant 3$. You want to find the third largest integer.

Show that this problem is in $P$.

**Solution:**

```
INPUT A[n]
For i = 1 to 3
        max = A[1]
        For j = 2 to n
                If A[j] > max
                        max = A[j]
                        index = j
                EndIf
        EndFor
```

$A[\text{index}] = 0$
EndFor
Return max

This algorithm uses $3n$ steps therefore it is $\Theta(n)$. So it is in $P$.

# Week 13– P and NP

**Time Complexity**

Let $M$ be a deterministic Turing Machine that halts on all inputs. The running time or time complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that $M$ uses on any input of length $n$.
(Big Oh notation, small Oh notation)
Let $t : \mathbb{N} \to \mathbb{R}$ be a function. Define the time complexity class, TIME$(t(n))$, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

**Example:**
Take the language $A = \{0^n 1^n \mid n \geqslant 0\}$. Obviously $A$ is a decidable language. How much time does a single-tape Turing machine need to decide $A$?

$M_1$: (Erase two symbols at each sweep)

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat :
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If neither 0s nor 1s remain on the tape, ACCEPT.
5. If there are some 1's but no 0's, or the opposite, REJECT.

$M_2$: (Reduce to half at each sweep)

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat :
3. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, REJECT.
4. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
5. If no 0s and no 1s remain on the tape, ACCEPT.
6. If there are some 1's but no 0's, or the opposite, REJECT.

These two machines decide the same language, but their efficiency is different. $M_1$ is $O(n^2)$ but $M_2$ is $O(n \log n)$.
If we use a two-tape machine $M_3$, we will sweep just once. Therefore the complexity will be even better. It will be $O(n)$. This is the best we can do, because just reading the input is $O(n)$.

**The Class P**

For our purposes, polynomial differences in running time are considered to be small, whereas exponential differences are considered to be large.
All reasonable deterministic computational models are polynomially equivalent.
$P$ is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

$$P = \bigcup_k \text{TIME}(n^k)$$

The class P plays a central role in our theory and is important because

1. P is invariant for all models of computation that are polynomially equivalent to the deterministic single-tape Turing machine, and
2. P roughly corresponds to the class of problems that are realistically solvable on a computer.

**Exercise 13-1:** A directed graph $G$ contains nodes $s$ and $t$. The PATH problem is to determine whether a directed path exists from $s$ to $t$.
Is PATH $\in P$?

**Solution:** YES
A brute-force algorithm that checks all possible paths does roughly $n^n$ operations. (Unacceptably high)
We use a breadth−first search. On input $\langle G, s, t\rangle$ where $G$ is a directed graph with nodes $s$ and $t$:

1. Place a mark on node $s$.
2. Repeat the following until no additional nodes are marked:
3. Scan all the edges of $G$. If an edge $(a, b)$ is found going from a marked node $a$ to an unmarked node $b$, mark $b$.
4. If $t$ is marked, accept. Otherwise, reject.

This algorithm for PATH is of polynomial time. ($O(n^3)$)

**Exercise 13-2:** A Hamiltonian path in a directed graph $G$ is a directed path that goes through each node exactly once. We consider the problem of testing whether a directed graph contains a Hamiltonian path connecting two specified nodes. Define:

$$\text{HAMPATH} = \{\langle G, s, t\rangle \mid \text{G is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

We can find a brute-force exponential time algorithm for the HAMPATH problem. Is there a polynomial time solution? Nobody knows.

### Verifiability

Even though we don't know of a fast (i.e., polynomial time) way to determine whether a graph contains a Hamiltonian path, if such a path were discovered we could easily check its correctness. Another polynomially verifiable problem is compositeness. Recall that a natural number is composite if it is the product of two integers greater than 1.
Some problems may not be polynomially verifiable. For example, take $\overline{\text{HAMPATH}}$, the complement of the HAMPATH problem. Even if we could determine (somehow) that a graph did not have a Hamiltonian path, we don't know of a way for someone else to verify its nonexistence without using the same exponential time algorithm for making the determination in the first place.
A verifier for a language $A$ is an algorithm $V$ where

$$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}$$

### The Class NP

We can characterize NP in two different, but equivalent ways:

- NP is the class of languages that have polynomial time verifiers.
- NP is the class of languages decided by some nondeterministic polynomial time Turing machine.

**Exercise 13-3:** A clique in an undirected graph is a subgraph, wherein every two nodes are connected by an edge. A $k-$clique is a clique that contains $k$ nodes. Define:

$$\text{CLIQUE} = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k-\text{clique}\}$$

Is CLIQUE in NP?

**Exercise 13-4:** We have a set of numbers $S = \{x_1, \ldots, x_k\}$ and a target number $N$. We want to determine whether the set has a subset that adds up to $N$.

$$\text{SUBSET-SUM} = \{\langle S, N \rangle \mid \text{ For some } \{y_1, \ldots, y_m\} \subseteq S, \text{ we have } \sum y_i = N\}$$

Note that $\{x_1, \ldots, x_k\}$ and $\{y_1, \ldots, y_m\}$ are considered to be multisets and so allow repetition of elements.
Is SUBSET-SUM in NP?

**Exercise 13-5:** If $G$ is an undirected graph, a vertex cover of $G$ is a subset of the nodes where every edge of $G$ touches one of those nodes. The vertex cover problem asks whether a graph contains a vertex cover of a specified size:

$$\text{VERTEX-COVER } = \{\langle G, k \rangle \mid G \text{ is an undirected graph that has a } k-\text{node vertex cover}\}$$

Is VERTEX-COVER in NP?

**Exercise 13-6:** TS (Traveling Salesman) PROBLEM: Given $n$ cities, and distances $d(i, j)$ between them, what is the minimum distance of a path that visits each city once?

HAMPATH (Hamiltonian Path) PROBLEM: Given a graph $G$, is there a path that goes through each node exactly once?

**a)** Reformulate the TS problem as a Yes/No problem. Call it TS2.

**b)** Given that the HAMPATH problem is NP-complete, show that TS2 is NP-complete.

**Solution:** **a)** TS2 (Traveling Salesman) PROBLEM: Given $n$ cities, and distances $d(i, j)$ between them, and a positive number $k$, is there a path that visits each city once and has length $\leqslant k$?

**b)** We have to show that

$$\text{HAMPATH} \leqslant_P \text{TS2}$$

Suppose we can solve any TS2 problem. Given a HAMPATH problem, transform it into TS2 as follows:

- If there is a connection between vertex $i$ and vertex $j$, set $d(i, j) = 1$.
- If there is no connection between vertex $i$ and vertex $j$, set $d(i, j) = 10$.
- Set $k = n$ (number of vertices)

If TS2 finds a path of length $\leqslant n$ (actually, we have length $= n$) it is the one we are looking for. Clearly, this reduction is of polynomial type.

**Exercise 13-7:** Consider the following problem:
*Given a graph, is there a way to partition the vertices into 4 subsets such that no two elements in a subset are connected?*
Show that this problem is in NP.

**Solution:** Suppose there are $n$ vertices. Then, there are at most $\frac{n(n-1)}{2} = \Theta(n^2)$ edges.

We can check a given solution by checking all edges one by one. (We will return FALSE if the two vertices connected by the edge belong to the same subset)

Therefore a given a solution can be verified in $\Theta(n^2)$ operations. The problem is in NP.

**Exercise 13-8:**
The SHORTEST PATH problem is defined as follows:
You are given a weighted undirected graph $G$, and nodes $s$ and $t$ in the graph and a number $W$. Is there a path from $s$ to $t$ with weight $\leqslant W$?

Show that this problem is in NP.

**Solution:**
A given solution contains $n-1$ edges. We can check each edge in $n-1$ steps if each vertex has its list of edges. Then we have to check the sum. Therefore we do $(n-1)^2 + n - 1$ operations, so verifying algorithm is $\Theta(n^2)$.
This is polynomial, so the problem is in NP.

**Exercise 13-9:**
The MINIMUM SPANNING TREE problem is defined as follows:
You are given a weighted undirected graph $G$ and a number $W$. Is there a spanning tree with weight $\leqslant W$?

Show that this problem is in NP.

**Solution:**
A given solution contains $\Theta(n)$ edges. We can check each edge in $\Theta(n)$ steps if each vertex has its list of edges. (If we have an unsorted list of edges, this will require $n^2$ operations but still, it is polynomial) Then we have to check the sum and also we have to check that all vertices are connected.
Verifying algorithm is $\Theta(n^2)$.
This is polynomial, so the problem is in NP.

# Week 14– NP Completeness

### P versus NP

We can summarize our results as:

- $P =$ the class of languages for which membership can be decided quickly.
- $NP =$ the class of languages for which membership can be verified quickly.

Is $P = NP$? Or is $P \subsetneq NP$? We don't know.

The question of whether $P = NP$ is one of the greatest unsolved problems in theoretical computer science and contemporary mathematics.
If these classes were equal, any polynomially verifiable problem would be polynomially decidable. Most researchers believe that the two classes are not equal because people have invested enormous effort to find polynomial time algorithms for certain problems in NP, without success.

### Polynomial Time Reducibility

A language $A$ is said to be polynomial time reducible to a language $B$ if the map reduction between them can be computed in polynomial time, denoted by $A \leqslant_p B$
This is, the time complexity of the Turing machine that computes the reduction mapping is bounded above by a polynomial function in the length of the input string.

**Theorem:** If $A \leqslant_p B$ and $B \in P$, then $A \in P$.

### NP Complete Languages

There are problems in NP whose complexity is related to that of the entire class. If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable. These problems are called NP-complete.
A language $B$ is NP-complete if it satisfies two conditions:

1. $B$ is in NP, and
2. Every $A$ in NP is polynomial time reducible to $B$.

**Theorem:** If $B$ is NP-complete and $B \in P$, then $P = NP$.

**Theorem:** If $B$ is NP-complete and $B \leqslant_P C$ for $C$ in NP, then $C$ is NP-complete.

### SAT Problem

The Boolean operations are: AND, OR, and NOT. A Boolean formula is satisfiable if some assignment of 0s and 1s to the variables makes the formula evaluate to 1. The satisfiability problem is to test whether a Boolean formula is satisfiable. Let

$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

**Theorem:** SAT is NP-complete.

### Examples of NP-Complete Problems

**Theorem:** CLIQUE is NP-complete.

**Theorem:** HAMPATH is NP-complete.

**Theorem:** SUBSET-SUM is NP-complete.

**Theorem:** VERTEX-COVER is NP-complete.

**Exercise 14-1:** The VERTEX COVER problem is defined as:

Given a graph and an integer $k$, is there a collection of $k$ vertices such that each edge is connected to one of the vertices in the collection?

Is VERTEX COVER in NP?

**Solution:** Given a solution, we can verify it in polynomial time by the following method:

INPUT A total of $n$ vertices, a list of $k$ vertices (cover), a list of $\ell$ edges.
For $i = 1$ to $k$
    For $j = 1$ to $\ell$
        If vertex $i$ is connected to edge $j$
            Mark edge $j$
        EndIf
    EndFor
EndFor
For $j = 1$ to $\ell$
    If edge $j$ is unmarked
        Return REJECT
    EndIf
EndFor
Return ACCEPT

We know that $k \leqslant n$ and $\ell \leqslant n(n-1)/2$ so this algorithm takes $O(n^3) + O(n^2) = O(n^3)$ time, i.e. polynomial time.

Therefore VERTEX COVER is in NP.

Note: We assume connection between vertices can be checked in $O(1)$ operation.

**Exercise 14-2:** The COLORING problem is defined as:

Given a graph and an integer $k$, is there a way to color the vertices with $k$ colors such that adjacent vertices are colored differently?

Is COLORING in NP?

**Solution:** Given a solution, we can verify it in polynomial time by the following method:

INPUT A list of $n$ vertices with each one having one of $k$ colors, a list of $\ell$ edges.
For $i = 1$ to $n - 1$
    For $j = i + 1$ to $n$
        If vertex $i$ is connected to vertex $j$ AND Color(i)==Color(j)
            Return REJECT
        EndIf
    EndFor
EndFor
Return ACCEPT

This algorithm takes $O(n^2)$ time, i.e. polynomial time.

Therefore COLORING is in NP.

Note: We assume connection between vertices can be checked in $O(1)$ operation.

**Exercise 14-3:** INDEPENDENT SET problem: Given a graph $G$ and an integer $k$, is there a subset $S$ of $k$ vertices such that no two vertices in $S$ are connected? (All vertices are independent)

VERTEX COVER problem: Given a graph $G$ and an integer $k$, is there a subset $S$ of $k$ vertices such that every edge has at least one endpoint in $S$?

Show that

$$\text{INDEPENDENT SET} \leqslant_P \text{VERTEX COVER}$$

**Solution:** Suppose we have an algorithm that decides VERTEX COVER in polynomial time.

INPUT A graph $G$ with $n$ vertices, an integer $k$.

VERTEXCOVER$(G, n - k)$ // If reject, reject, if accept, accept.

(Note that, if there is a vertex cover of $n - k$ elements, the remaining $k$ elements form an independent set. )

**Exercise 14-4:** PARTITION Problem: Given a finite set of integers, determine if it can be partitioned into two sets such that the sum of all integers in the first set equals the sum of all integers in the second set.

SUBSET SUM Problem: Given a set of integers and a number $k$, determine if there is a subset of these integers whose sum is $k$.

Show that

$$\text{PARTITION} \leqslant_P \text{SUBSET SUM}$$

**Solution:** Suppose we have an algorithm that decides SUBSET SUM in polynomial time.

INPUT A set $S$ of integers, an integer $k$.

Add all elements of $S$. Call the result TOTAL.

If TOTAL is odd

    Return REJECT

else

    SUBSETSUM$(S, TOTAL/2)$  // If reject, reject, if accept, accept.

EndIf