A Concise Note on Theory of Computation (TOC)

Samena Bahleri
samenabahleri09@gmail.com

November 11, 2025

This page left intentionally blank

Contents

1 Introduction

1.1 TOC Development History

1. **Alan Turing (1936)**

   Introduced the concept of the Turing Machine, providing a formal model of computation and establishing the foundation for decidability, computability, and the limits of what machines can compute.

2. **Warren McCulloch and Walter Pitts (1943)**

   Proposed a mathematical model of artificial neurons, linking logic, computation, and brain function, and laying groundwork for automata theory and neural networks.

3. **Stephen Kleene (1943–1956)**

   Developed the theory of regular expressions and recursive functions, formalizing the foundations of computability and influencing automata and language theory.

4. **Noam Chomsky (1956)**

   Introduced the Chomsky hierarchy of formal grammars, classifying languages based on their generative power and deeply connecting syntax with automata theory.

5. **John Backus (1959)**

   Created the Backus–Naur Form (BNF) for describing the syntax of programming languages, bridging formal language theory with practical computing.

6. **Stephen Cook (1971)**

   Defined the concept of NP-completeness, introducing the Cook–Levin theorem, which became central to computational complexity theory.

7. **Richard Karp (1972)**

   Expanded on Cook's work by identifying 21 NP-complete problems, solidifying the framework for studying computational intractability.

1.2 Central Concepts of TOC

1. Alphabets

   Example:

   (a) $\Sigma = \{0, 1\}$ (Binary Alphabet)

   (b) $\Sigma = \{a, b, c\}$ (Ternary Alphabet)

   (c) $\Sigma = \{a, b, c, \dots, z\}$ (English Alphabet)

   (d) $\Sigma = \{0, 1, 2, \dots, 9\}$ (Decimal Digits)

2. Strings

   Example:

   (a) Over $\Sigma = \{0, 1\}$: "101", "0001", "1110"
   (b) Over $\Sigma = \{a, b, c\}$: "abc", "aab", "ccba"
   (c) Over $\Sigma = \{0, 1, 2, \ldots, 9\}$: "123", "4567", "890"
   (d) Empty String: Denoted by ε, represents a string with no characters.
   (e) length of string: For a string S, the length is denoted by $|S|$. For example, if $S =$ "11011", then $|S| = 5$.

3. Powers

   Example:

   (a) $\Sigma = \{0, 1\}$
      i. $\Sigma^0 = \{\varepsilon\}$
      ii. $\Sigma^1 = \{0, 1\}$
      iii. $\Sigma^2 = \{00, 01, 10, 11\}$
      iv. $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$
   (b) $\Sigma^* = \bigcup_{k \geq 0} \Sigma^k$

4. Concatenation

   Example:

   (a) Let $s_1 =$ "101" and $s_2 =$ "11" over $\Sigma = \{0, 1\}$. Then, $s_1 \cdot s_2 =$ "10111".
   (b) Let $s_1 =$ "*ab*" and $s_2 =$ "*cde*" over $\Sigma = \{a, b, c, d, e\}$. Then, $s_1 \cdot s_2 =$ "*abcde*".

5. Language

   Example:

   (a) $L = \{w \mid w \text{ has an even number of 1s}\}$
   (b) $L = \{w \mid w \text{ starts with 'a' and ends with 'b'}\}$
   (c) $L = \{0^i 1^j \mid 0 \geq i \geq j\}$, Hence:
      – L contains strings like " ", "01", "0011", "000111", etc.
      – L does not contain strings like "10", "1100", "0110", etc.

2 Regular Languages

Regular Languages are a class of formal languages that can be recognized by finite automata, specifically Deterministic Finite Automata (DFA) and Non-deterministic Finite Automata (NFA). They are called "regular" because they can be described using regular expressions, which provide a concise way to represent patterns in strings.

2.1 Regular Expressions

A Regular Expression (regex or regexp) is a sequence of characters that defines a search pattern, primarily used for string matching within texts. Regular expressions are built using a combination of symbols and operators to represent sets of strings.

1. Union

   Symbolic notation for union is:

   $$L + M = \{x \in L \cup y \in M\}$$

   Example:

   Let L, M be two languages over the alphabet $\Sigma = \{1, 0\}$ such that:

   (a) $L = \{1^n \mid n \geq 1\}$
   (b) $M = \{0^n \mid n \geq 1\}$

   Then, the union of L and M is:

   $$L + M = \{1^n \mid n \geq 1\} \cup \{0^n \mid n \geq 1\}$$

2. Concatenation

   Symbolic notation for concatenation is:

   $$L \cdot M = \{x \in L \times y \in M\}$$

   Example:

   Let L, M be two languages over the alphabet $\Sigma = \{1, 0\}$ such that:

   (a) $L = \{1^n \mid n \geq 1\}$
   (b) $M = \{0^n \mid n \geq 1\}$

   Then, the concatenation of L and M is:

   $$L \cdot M = \{1^n 0^m \mid n \geq 1, m \geq 1\}$$

   In detail:

   Suppose $L = \{1, 11\}$ and $M = \{0, 00\}$. Then,

   $$L \cdot M = \{10, 100, 110, 1100\}.$$

3. Kleene Star

   Symbolic notation for Kleene Star is:

   $$L^* = \bigcup_{k=0}^{\infty} L^k$$

where $L^0 = \{\varepsilon\}$ and $L^k = L \cdot L^{k-1}$ for $k \geq 1$.

Example 1:

Let L be a language over the alphabet $\Sigma = \{0, 1\}$, and suppose $L = \{0, 1\}$. Then

(a) $L^0 = \{\varepsilon\}$

(b) $L^1 = \{0, 1\}$

(c) $L^2 = L \cdot L = \{00, 01, 10, 11\}$

(d) $L^3 = L \cdot L^2 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

(e) $L^4 = L \cdot L^3 = \{0000, 0001, 0010, 0011, 0100, 0101, \ldots\}$

Example 2:

Let L be a language over the alphabet $\Sigma = \{0, 1\}$ such that:

(a) $0^*1^* = \{\varepsilon, 1, 11, 111, \ldots, 0, 01, 001, 0001, \ldots, 00, 000, 0000, \ldots\}$

(b) $1^*0 = 1^* \cdot 0 = \{0, 10, 110, 1110, 11110, \ldots\}$

(c) $10^* = \{1, 10, 100, 1000, 10000, \ldots\}$

(d) $1 + 01^* = \{1\} \cup \{0, 01, 011, 0111, \ldots\}$

(e) $1^* \cup \{1^*0\} = \{\varepsilon, 1, 11, 111, \ldots\} \cup \{0, 10, 110, 1110, 11110, \ldots\}$

2.2 Unix RegExp

In Unix, regular expressions are used in various command-line tools for pattern matching and text processing.

Example:

1. `[bch]at` matches `bat`, `cat`, and `hat`.
2. `[10xyz]ab` matches `1ab`, `0ab`, `xab`, `yab`, and `zab`.

2.3 Identities of Regular Expressions

1. Basic identities
$$f + \varnothing = f$$
$$f \cdot \varepsilon = \varepsilon \cdot f = f$$
$$f \cdot \varnothing = \varnothing \cdot f = \varnothing$$
2. Idempotent law
$$f + f = f$$
3. Commutative law (union only)
$$f + g = g + f$$
4. Associative laws
$$(f + g) + f = f + (g + f)$$
$$(f \cdot g) \cdot f = f \cdot (g \cdot f)$$

5. Distributive laws
$$f \cdot (g + f) = f \cdot g + f \cdot f$$
$$(f + g) \cdot f = f \cdot f + g \cdot f$$

6. Kleene star identities
$$\begin{aligned} f^* &= \varepsilon + ff^* \\ f^* &= \varepsilon + f^*f \\ (f^*)^* &= f^* \\ \varnothing^* &= \varepsilon \\ \varepsilon^* &= \varepsilon \end{aligned}$$

7. Positive closure
$$f^+ = ff^* = f^*f$$

8. Absorption laws
$$f + ff^* = ff^*$$
$$f + f^*f = f^*f$$

9. Star concatenation collapse
$$f^*f^* = f^*$$

10. Star unfolding
$$\varepsilon + ff^* = \varepsilon + f^*f = f^*$$

11. Star shift (rotation law)
$$(fg)^*f = f(gf)^*$$

12. Union–star expansion
$$(f + g)^* = (f^*g^*)^*$$

3 Deterministic Finite Automata (DFA)

A DFA is a finite-state machine where every state has exactly one transition for each input symbol. Given an input and a current state, the next state is uniquely determined, DFA is formally defined as a 5-tuple $M = (W, \Sigma, V, S, F)$ where:

1. $W = \{w_1, w_2, w_3 \cdots w_n\}$, finite set of states.
2. $\Sigma = \{0, 1\}$, finite set of input symbols .
3. $V : W \times \Sigma \to W.E.g, V(x, 1) = y$, transition function that takes a state and an input symbol and returns the next state.
4. $S \in W$ is the start state where the computation begins.
5. $F \subseteq W$ is the set of accept (final) states.

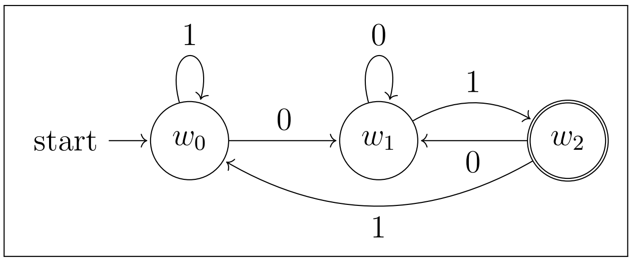
3.1 Transition Diagram

A graphical representation of the DFA, where states are represented as circles, and transitions are represented as arrows labeled with input symbols.

Example:

Let

$$L(M) = \{\, x \in \{0,1\}^* \mid x \text{ ends with } 01 \,\}$$



In this DFA:

1. States: $W = \{w_0, w_1, w_2\}$
2. Input Alphabet: $\Sigma = \{0, 1\}$
3. Transition Function V:

$$\begin{aligned} V(w_0,0) &= w_1 & V(w_1,1) &= w_2 \\ V(w_0,1) &= w_0 & V(w_2,0) &= w_1 \\ V(w_1,0) &= w_1 & V(w_2,1) &= w_0 \end{aligned}$$

4. Start State: $S = w_0$
5. Accept State: $F = w_2$

3.2 Transition Table

From that Transition Diagram we can construct a Transition Table as follow

W	**0**	**1**
w_0	w_1	w_0
w_1	w_1	w_2
w_2	w_1	w_0

3.3 Extended Transition Function

The extended transition function $\hat{V} : W \times \Sigma^* \to W$ is defined by

1. $\hat{V}(w, \varepsilon) = w$, $\forall w \in W$.
2. $\hat{V}(w, xy) = V\big(\hat{V}(w,x), y\big)$, $\forall w \in W$, $x \in \Sigma^*$, $y \in \Sigma$.

Hence, from the above DFA we compute (starting from the start state w_0):

1. $\hat{V}(w_0, \varepsilon) = w_0$
2. $\hat{V}(w_0, 0) = V(\hat{V}(w_0, \varepsilon), 0) = w_1$
3. $\hat{V}(w_0, 1) = V(\hat{V}(w_0, \varepsilon), 1) = w_0$
4. $\hat{V}(w_0, 01) = V(\hat{V}(w_0, 0), 1) = w_2$
5. $\hat{V}(w_0, 101) = V(\hat{V}(w_0, 10), 1) = w_2$.
6. $\hat{V}(w_0, 1101) = V(\hat{V}(w_0, 110), 1) = w_2$.
7. $\hat{V}(w_0, 100) = V(\hat{V}(w_0, 10), 0) = w_1$.

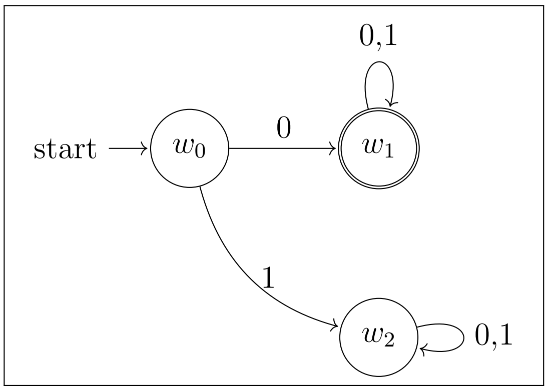
In general:

$$\hat{V}(w_0, x) = F \;\leftrightarrow\; x \text{ ends with } xy, \text{ so } L(M) = \{\, x \in \{0,1\}^* \mid \hat{V}(w_0, x) \in F \,\}.$$

Example 1:
Let

$$L(M_1) = \{0(0+1)^*\}$$

Hence



Specifically, it only accept the input strings that start with 0. Oterwise, it goes to the dead state w_2. Formally,

$$L(M_1) = \begin{cases} \{\, 0x \mid x \in \{0,1\}^* \,\}, & \text{if the first symbol is 0,} \\ \varnothing, & \text{otherwise.} \end{cases}$$

1. Transition Function V:

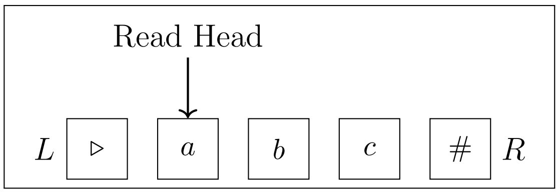
$$\begin{aligned} V(w_0,0)&=w_1 & V(w_1,1)&=w_1\\ V(w_0,1)&=w_2 & V(w_2,0)&=w_2\\ V(w_1,0)&=w_1 & V(w_2,1)&=w_2 \end{aligned}$$

2. Transition Table:

W	**0**	**1**
w_0	w_1	w_2
w_1	w_1	w_1
w_2	w_2	w_2

3.4 Two-way Finite Automata (2DFA)

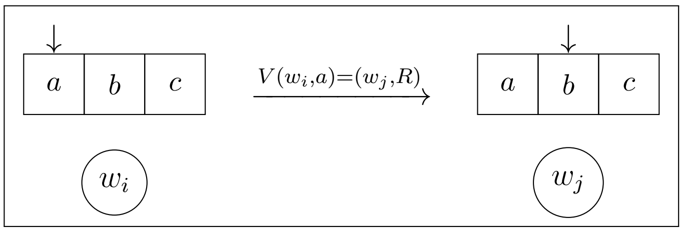
Formally, $V : W \times (\Sigma \cup \{\triangleright, \#\}) \to W \times \{L, R\}$ is the transition function, where $\#$ is the end-of-tape marker and $\triangleright$ is the start-of-tape marker.



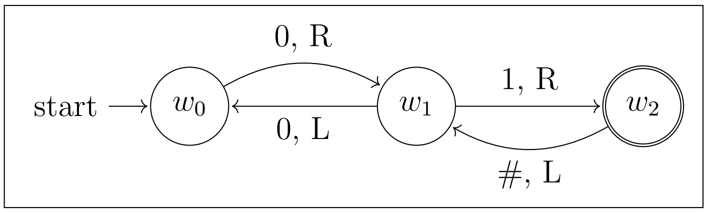
In general:

$$V(w_i, a) = (w_j, R)$$

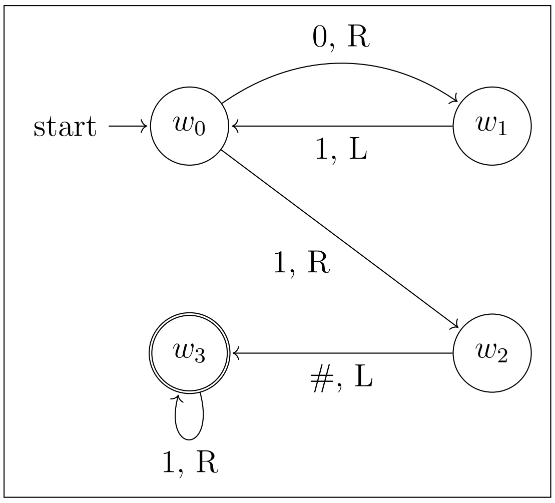
This means that when the 2DFA is in state w_i and reads the symbol a, it transitions to state w_j and moves the read head one cell to the right (R).



Intuitively, a 2DFA operates like a standard DFA



Example:
Let



This 2DFA accepts only the string 1. The transition function is defined as:

$$
\begin{aligned}
V(w_0,0) &= (w_1,R) & \qquad V(w_2,\#) &= (w_3,L)\\
V(w_0,1) &= (w_2,R) & \qquad V(w_3,1) &= (w_3,R)\\
V(w_1,1) &= (w_0,L) & &
\end{aligned}
$$

1. **Accepted:** Input 1
   (a) Start: w_0, $\boxed{\underline{1}}\boxed{\#}\boxed{\sqcup}\boxed{\sqcup}\boxed{\sqcup}\cdots$
   (b) Read 1, move R: w_2, $\boxed{1}\boxed{\underline{\#}}\boxed{\sqcup}\boxed{\sqcup}\boxed{\sqcup}\cdots$
   (c) Read #, move L: w_3, $\boxed{\underline{1}}\boxed{\#}\boxed{\sqcup}\boxed{\sqcup}\boxed{\sqcup}\cdots$
   (d) Read 1, move R: w_3, $\boxed{1}\boxed{\underline{\#}}\boxed{\sqcup}\boxed{\sqcup}\boxed{\sqcup}\cdots$
   (e) Accept (in state w_3 at end marker)

2. **Rejected (crash):** Input 0
   (a) Start: w_0, $\boxed{\underline{0}}\boxed{\#}\boxed{\sqcup}\boxed{\sqcup}\boxed{\sqcup}\cdots$
   (b) Read 0, move R: w_1, $\boxed{0}\boxed{\underline{1}}\boxed{\#}\boxed{\sqcup}\boxed{\sqcup}\cdots$
   (c) Read 1: No transition for $(w_1, 1)$, Crash

3. **Rejected (crash):** Input 11
   (a) Start: w_0, $\boxed{\underline{1}}\boxed{1}\boxed{\#}\boxed{\sqcup}\boxed{\sqcup}\cdots$
   (b) Read 1, move R: w_2, $\boxed{1}\boxed{\underline{1}}\boxed{\#}\boxed{\sqcup}\boxed{\sqcup}\cdots$
   (c) Read 1: No transition for $(w_2, 1)$, Crash

4. **Rejected (crash):** Input 10
   (a) Start: w_0, $\boxed{\underline{1}}\boxed{0}\boxed{\#}\boxed{\sqcup}\boxed{\sqcup}\cdots$
   (b) Read 1, move R: w_2, $\boxed{1}\boxed{\underline{0}}\boxed{\#}\boxed{\sqcup}\boxed{\sqcup}\cdots$

(c) Read 0: No transition for $(w_2, 0)$, Crash

5. **Infinite loop:** Input 01

   (a) Start: w_0, $\boxed{\underline{0}}\boxed{1}\boxed{\#}\boxed{\sqcup}\boxed{\sqcup}\cdots$

   (b) Read 0, move R: w_1, $\boxed{0}\boxed{\underline{1}}\boxed{\#}\boxed{\sqcup}\boxed{\sqcup}\cdots$

   (c) Read 1, move L: w_0, $\boxed{\underline{0}}\boxed{1}\boxed{\#}\boxed{\sqcup}\boxed{\sqcup}\cdots$

   (d) Read 0, move R: w_1, $\boxed{0}\boxed{\underline{1}}\boxed{\#}\boxed{\sqcup}\boxed{\sqcup}\cdots$

   (e) Read 1, move L: w_0, $\boxed{\underline{0}}\boxed{1}\boxed{\#}\boxed{\sqcup}\boxed{\sqcup}\cdots$

   (f) Cycles indefinitely between steps 3–5, Infinite loop

6. **Infinite loop:** Input 010101

   (a) Start: w_0, $\boxed{\underline{0}}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{\#}\cdots$

   (b) Read 0, move R: w_1, $\boxed{0}\boxed{\underline{1}}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{\#}\cdots$

   (c) Read 1, move L: w_0, $\boxed{\underline{0}}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{\#}\cdots$

   (d) Cycles between positions 0 and 1, Infinite loop

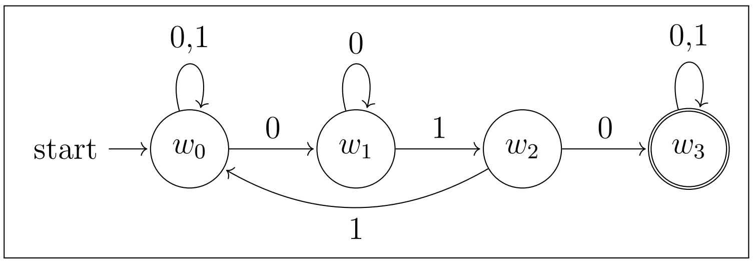
4 Non-deterministic Finite Automata (NFA)

An NFA is similar like DFA, but a state can have zero, one, or many possible transitions for the same input symbol, and it may even move without consuming input (ε-Transition). It's more flexible in structure, though it recognizes the same class of languages as a DFA. Formally, an NFA is defined as a 5-tuple $M = (W, \Sigma, V, S, F)$ where $V : W \times \Sigma \to 2^{|W|}$, and the language it recognizes is defined as: $L(M) = \{x \mid V(w, x) \cap F \neq \varnothing\}$.

Example:

Let

$$L(M) = \{\, x \in \{0,1\}^* \mid x \text{ contains the substring } 010 \,\}$$



In this NFA:

1. Transition Function V:

$$\begin{aligned} V(w_0,0) &= \{w_0,w_1\} & V(w_2,0) &= \{w_3\} \\ V(w_0,1) &= \{w_0\} & V(w_2,1) &= \{w_0\} \\ V(w_1,0) &= \{w_1\} & V(w_3,0) &= \{w_3\} \\ V(w_1,1) &= \{w_2\} & V(w_3,1) &= \{w_3\} \end{aligned}$$

2. Transition Table:

W	**0**	**1**
w_0	$\{w_0,w_1\}$	$\{w_0\}$
w_1	$\{w_1\}$	$\{w_2\}$
w_2	$\{w_3\}$	$\{w_0\}$
w_3	$\{w_3\}$	$\{w_3\}$

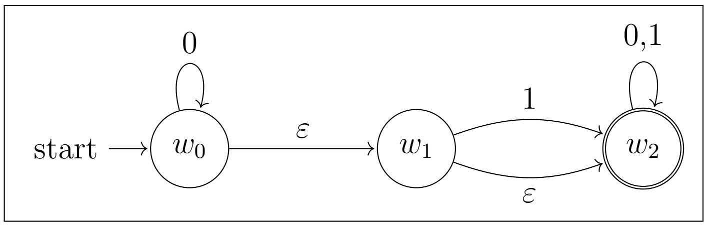
4.1 ε-NFA

An ε-NFA is an extension of the NFA that allows transitions without consuming any input symbols, known as ε-transitions. Formally, $V : W \times (\Sigma \cup \{\varepsilon\}) \to 2^{|W|}$.

Example:

Let

$$L(M) = \{\, x \in \{0,1\}^* \mid V^*(w_0,x) \cap \{w_2\} \neq \varnothing \,\}$$



In this ε-NFA:

$$\begin{aligned} V(w_0,\varepsilon) &= \{w_1\} & V(w_1,\varepsilon) &= \{w_2\} \\ V(w_0,0) &= \{w_0\} & V(w_2,0) &= \{w_2\} \\ V(w_1,1) &= \{w_2\} & V(w_2,1) &= \{w_2\} \end{aligned}$$

4.2 ε-Closure

The ε-closure of a state w is the set of all states reachable from w using only ε-transitions, including w itself. Formally,

$$\varepsilon\text{-closure}(w) = \{\, u \in W \mid \exists k \geq 0,\ w = w_0,\ w_k = u \text{ such that } (w_i,\varepsilon,w_{i+1}) \in V, \forall 0 \leq i < k \,\}.$$

For a set of states $S \subseteq W$,

$$\varepsilon\text{-closure}(S) = \bigcup_{w \in S} \varepsilon\text{-closure}(w).$$

In light of this, for the example ε-NFA above:

1. $\varepsilon(w_0) = \{w_0, w_1, w_2\}$ (since $w_0 \xrightarrow{\varepsilon} w_1 \xrightarrow{\varepsilon} w_2$)
2. $\varepsilon(w_1) = \{w_1, w_2\}$ (since $w_1 \xrightarrow{\varepsilon} w_2$)
3. $\varepsilon(w_2) = \{w_2\}$ (no ε-transitions from w_2)

Moreover, the extended Transition Function for ε-NFA is defined as:

1. $\hat{V}(w_0, \varepsilon) = \varepsilon(w_0) = \{w_0, w_1, w_2\}$
2. $\hat{V}(w_0, 0) = V(\varepsilon(w_0), 0) = V(\{w_0, w_1, w_2\}, 0) = \{w_0, w_2\}$
3. $\hat{V}(w_0, 1) = V(\varepsilon(w_0), 1) = V(\{w_0, w_1, w_2\}, 1) = \{w_2\}$

Thus, the ε-closures can be summarized in the following table:

W	**ε-Closure**
w_0	$\{w_0, w_1, w_2\}$
w_1	$\{w_1, w_2\}$
w_2	$\{w_2\}$

5 Moore and Mealy Machine

5.1 Moore Machine

Formally, a Moore machine is defined as a 6-tuple

$$M = (W, \Sigma, \Delta, V, \lambda, S)$$

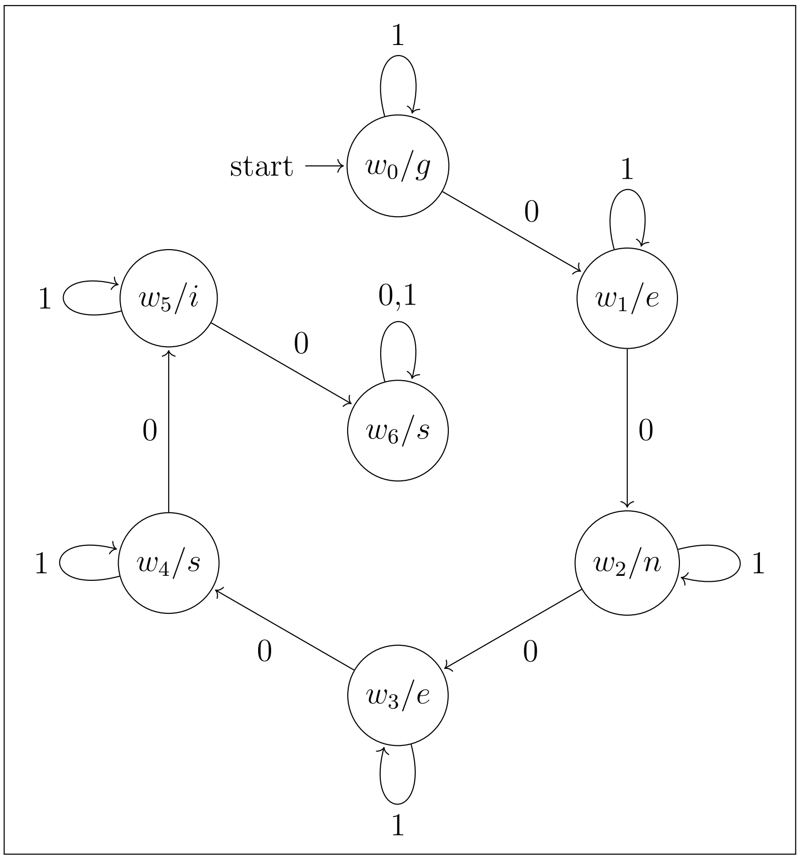
where

1. Δ is the output alphabet,
2. $\lambda : W \to \Delta$ is the output function that maps each state to an output symbol.

Example:
Let $W = \{w_0, w_1, w_2, w_3, w_4, w_5, w_6\}$, $\Sigma = \{0, 1\}$, $\Delta = \{g, e, n, s, i\}$, and $S = w_0$.

1. Transition Function V and Output Function λ:

W	**0**	**1**
w_0	w_1	w_0
w_1	w_2	w_1
w_2	w_3	w_2
w_3	w_4	w_3
w_4	w_5	w_4
w_5	w_6	w_5
w_6	w_6	w_6



Operation on input string 000000:

Input	**State**	**Output**
–	w_0	g
0	w_1	e
0	w_2	n
0	w_3	e
0	w_4	s
0	w_5	i
0	w_6	s

Hence, final output sequence: $\boxed{genesis}$

5.2 Mealy Machine

Bassically, a Mealy machine is defined as a 6-tuple, just like the Moore machine:

$$M = (W, \Sigma, \Delta, V, \lambda, S)$$

In this case, if Moore machine's output function maps states to outputs, Mealy machine's output function maps state-input pairs to outputs:

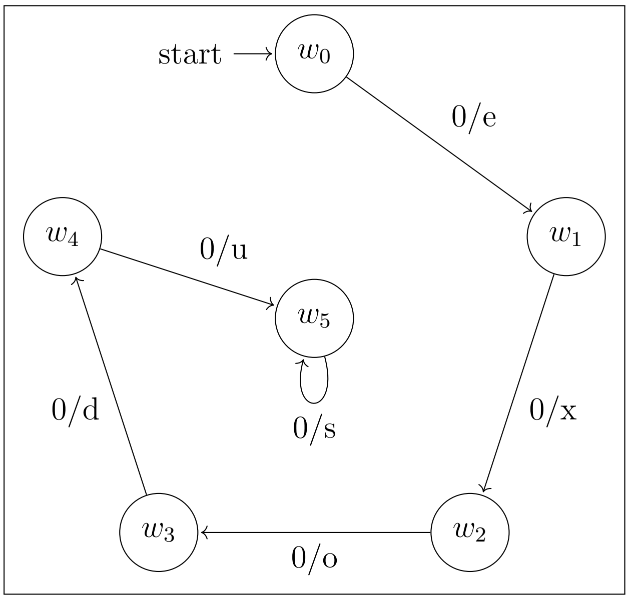
1. Δ is the output alphabet,
2. $\lambda : W \times \Sigma \to \Delta$ is the output function that maps each state-input pair to an output symbol,

3. The output is determined by the transition: $\lambda(w,x) = y$, where $w \in W$, $x \in \Sigma$, and $y \in \Delta$.

Example:
Let $W = \{w_0, w_1, w_2, w_3, w_4, w_5\}$, $\Sigma = \{0,1\}$, $\Delta = \{e, x, o, d, u, s\}$, and $S = w_0$.

1. Transition Function V and Output Function λ:

$$\begin{aligned}
V(w_0,0) &= w_1, & \lambda(w_0,0) &= e,\\
V(w_1,0) &= w_2, & \lambda(w_1,0) &= x,\\
V(w_2,0) &= w_3, & \lambda(w_2,0) &= o,\\
V(w_3,0) &= w_4, & \lambda(w_3,0) &= d,\\
V(w_4,0) &= w_5, & \lambda(w_4,0) &= u,\\
V(w_5,0) &= w_5, & \lambda(w_5,0) &= s.
\end{aligned}$$



Operation on input string 000000:

Input	State	Output	Transition
–	w_0	–	–
0	w_1	e	$V(w_0,0) = w_1, \lambda(w_0,0) = e$
0	w_2	x	$V(w_1,0) = w_2, \lambda(w_1,0) = x$
0	w_3	o	$V(w_2,0) = w_3, \lambda(w_2,0) = o$
0	w_4	d	$V(w_3,0) = w_4, \lambda(w_3,0) = d$
0	w_5	u	$V(w_4,0) = w_5, \lambda(w_4,0) = u$
0	w_5	s	$V(w_5,0) = w_5, \lambda(w_5,0) = s$

Hence, final output sequence: $\boxed{exodus}$

6 Arden's Theorem

Let f, g, h be regular expressions over an alphabet Σ.

If

$$f = g + fh$$

and

$$\varepsilon \notin L(h),$$

then the equation has a *unique solution*, namely

$$f = gh^*.$$

Proof.

Assume that

$$f = g + fh$$

with $\varepsilon \notin L(h)$.

We first show that $f = gh^*$ is a solution.

$$\begin{aligned} gh^* &= g(\varepsilon + hh^*) \\ &= g + ghh^* \\ &= g + (gh^*)h. \end{aligned}$$

Thus,

$$f = gh^*$$

satisfies the equation $f = g + fh$.

Next, we prove uniqueness.

Let f be any solution of the equation. By repeated substitution,

$$\begin{aligned} f &= g + fh \\ &= g + (g + fh)h \\ &= g + gh + fh^2 \\ &= g + gh + gh^2 + \cdots \end{aligned}$$

Hence,

$$f = g(\varepsilon + h + h^2 + h^3 + \cdots) = gh^*.$$

Since $\varepsilon \notin L(h)$, the expression h^* is well-defined and the solution is unique.

Therefore, the unique solution of

$$f = g + fh$$

is

$$\boxed{f = gh^*}$$

QED

7 Conversion

7.1 NFA to DFA

1. Given an NFA

$$N = (W, \Sigma, V, S, F)$$

2. the equivalent DFA

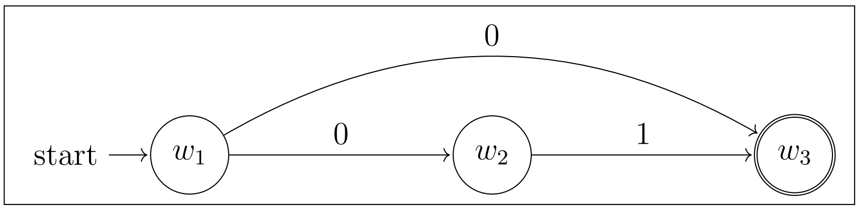
$$D = (W', \Sigma, V', S', F')$$

where

$$W' = 2^{|W|} \qquad V'(W, x) = \varepsilon\text{-closure}\left(\bigcup_{w \in W} V(w, x)\right)$$
$$S' = \varepsilon\text{-closure}(S) \qquad F' = \{\, W \in W' \mid W \cap F \neq \varnothing \,\}$$

In general

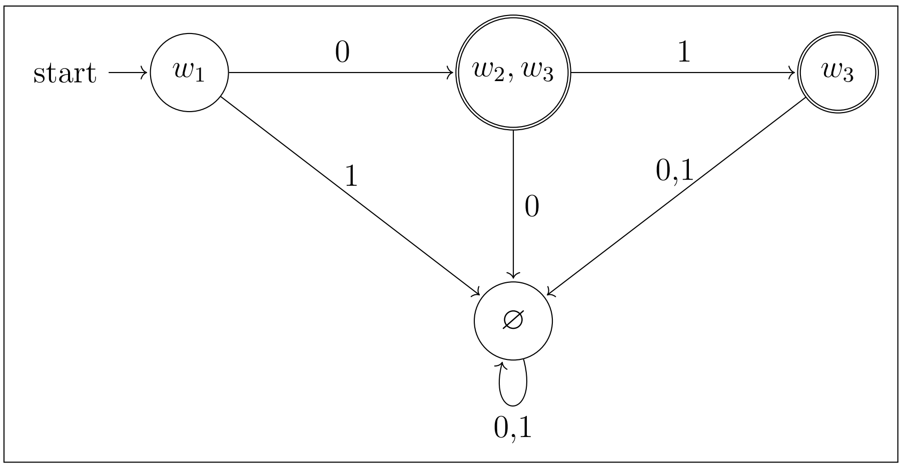
1. NFA



NFA Transition Table

W	**0**	**1**
w_1	$\{w_2, w_3\}$	$\varnothing$
w_2	$\varnothing$	$\{w_3\}$
w_3	$\varnothing$	$\varnothing$

2. DFA



DFA Transition Table

W'	**0**	**1**
w_1	$\{w_2, w_3\}$	$\varnothing$
$\{w_2, w_3\}$	$\varnothing$	$\{w_3\}$
$\{w_3\}$	$\varnothing$	$\varnothing$
$\varnothing$	$\varnothing$	$\varnothing$

1. **Example 1:**

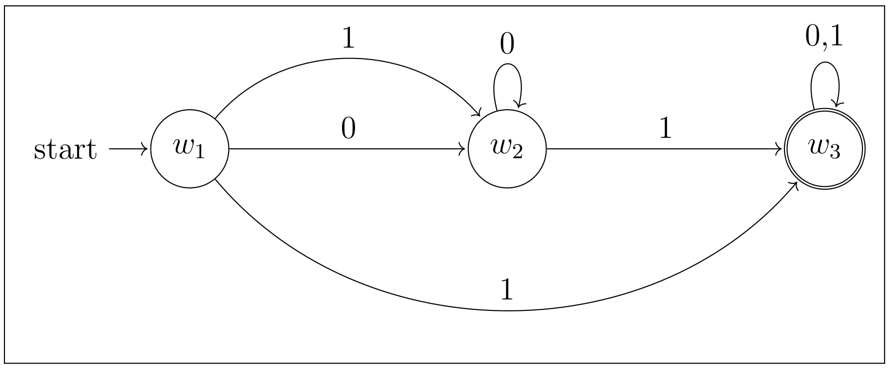
   Consider the NFA, where

$$W = \{w_1, w_2, w_3\}, \quad \Sigma = \{0, 1\}, \quad F = \{w_3\},$$

and the transition function V is defined as:

$$\begin{aligned} V(w_1, 0) &= \{w_2\}, & V(w_2, 0) &= \{w_2\}, \\ V(w_1, 1) &= \{w_2, w_3\}, & V(w_3, 0) &= \{w_3\}, \\ V(w_2, 1) &= \{w_3\}, & V(w_3, 1) &= \{w_3\}. \end{aligned}$$

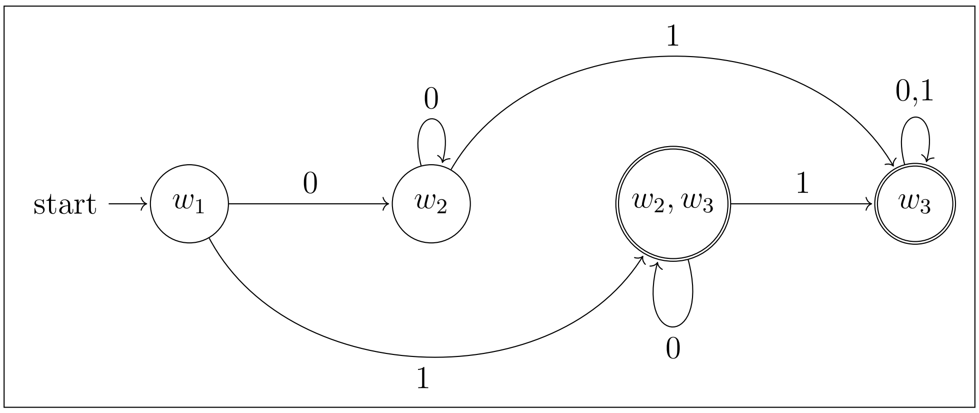
(a) NFA



Transition Table

W	**0**	**1**
w_1	w_2	$\{w_2, w_3\}$
w_2	w_2	w_3
w_3	w_3	w_3

(b) DFA



DFA Transition Table

W	**0**	**1**
w_1	w_2	$\{w_2, w_3\}$
w_2	w_2	w_3
w_3	w_3	w_3
$\{w_2, w_3\}$	$\{w_2, w_3\}$	w_3

2. **Example 2:**

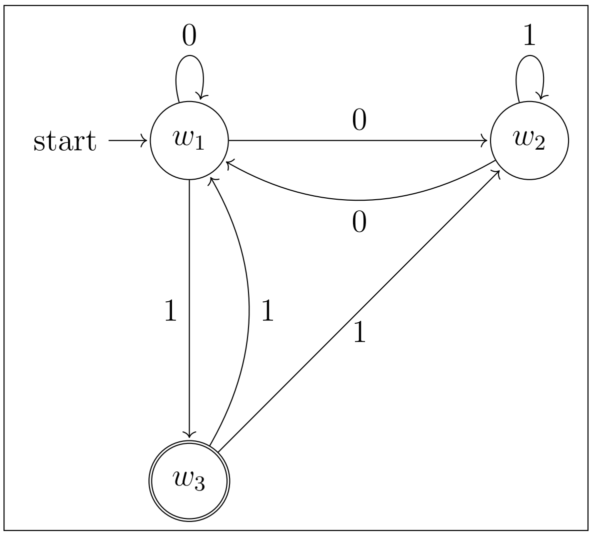
Consider the NFA:

$$W = \{w_1, w_2, w_3\}, \qquad \Sigma = \{0, 1\}, \qquad F = \{w_3\},$$

with transition function

$$V(w_1, 0) = \{w_1, w_2\}, \qquad V(w_2, 0) = \{w_1\}, \qquad V(w_3, 0) = \varnothing,$$
$$V(w_1, 1) = \{w_3\}, \qquad V(w_2, 1) = \{w_2\}, \qquad V(w_3, 1) = \{w_1, w_2\}.$$

(a) NFA



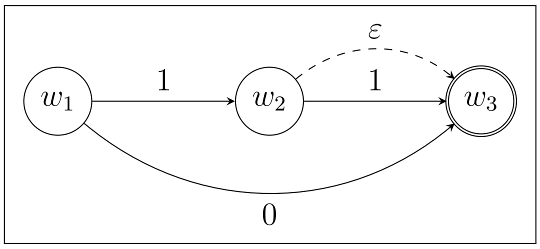
(b) DFA Transition Table

W	**0**	**1**
w_1	$\{w_1, w_2\}$	w_3
$\{w_1, w_2\}$	$\{w_1, w_2\}$	$\{w_2, w_3\}$
w_3	$\varnothing$	$\{w_1, w_2\}$
$\{w_2, w_3\}$	w_1	$\{w_1, w_2\}$

7.2 ε-NFA to NFA

Let

$$W = \{w_1, w_2, w_3\}, \quad \Sigma = \{0, 1\}, \quad F = \{w_3\}$$



1. Transition function of the ε-NFA:

W	**0**	**1**	ε
w_1	$\{w_3\}$	$\{w_2\}$	$\varnothing$
w_2	$\varnothing$	$\{w_3\}$	$\{w_3\}$
w_3	$\varnothing$	$\varnothing$	$\varnothing$

2. ε-closures:

$$\varepsilon(w_1) = \{w_1\}, \quad \varepsilon(w_2) = \{w_2, w_3\}, \quad \varepsilon(w_3) = \{w_3\}$$

3. Transition function of the equivalent NFA:

$$V_N(w,x) = \varepsilon\text{-closure}\Big(\bigcup_{w' \in \varepsilon(w)} V(w',x)\Big)$$

W	**0**	**1**
w_1	$\{w_3\}$	$\{w_2, w_3\}$
w_2	$\varnothing$	$\{w_3\}$
w_3	$\varnothing$	$\varnothing$

4. Accepting states of the NFA:

$$F_N = \{w \in W \mid \varepsilon(w) \cap F \neq \varnothing\} = \{w_2, w_3\}$$

7.3 NFA to RegExp

To convert an NFA to a regular expression, we follow these steps:

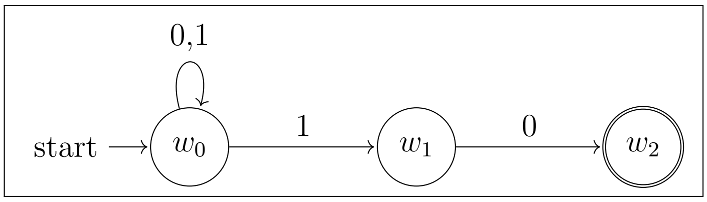
1. Represent each transition as a regular expression. If multiple transitions exist between the same states, combine them with + (union).
2. Identify the start and accepting states.
3. Iteratively eliminate intermediate states:
   - For a state w_k with incoming transitions from w_i and outgoing transitions to w_j, replace paths through w_k using the formula:
     $$R_{ij}^{\text{new}} = R_{ij}^{\text{old}} + R_{ik}(R_{kk})^* R_{kj}$$
     where R_{ik} is the regex from w_i to w_k, R_{kk} is the loop at w_k, and R_{kj} is the regex from w_k to w_j.
4. After all intermediate states are removed, the resulting regex from the start to accepting state is the regular expression of the language.

Example:
Let the NFA be:

$$W = \{w_0, w_1, w_2\}, \quad \Sigma = \{0,1\}, \quad S = \{w_0\}, \quad F = \{w_2\}$$

Transition function V:

$$\begin{aligned} V(w_0,0) &= \{w_0\}, & V(w_0,1) &= \{w_0, w_1\} \\ V(w_1,0) &= \{w_2\}, & V(w_1,1) &= \varnothing \\ V(w_2,0) &= \varnothing, & V(w_2,1) &= \varnothing \end{aligned}$$



1. Step 1: Initial Regular Expressions

   We label each transition with its corresponding regular expression:

$$\begin{aligned}
&w_0 \xrightarrow{0+1} w_0 \quad \text{(self-loop)} \\
&w_0 \xrightarrow{1} w_1 \\
&w_1 \xrightarrow{0} w_2 \\
&w_2 \text{ has no outgoing transitions}
\end{aligned}$$

   Initial edge labels:

$$\begin{aligned}
R_{w_0,w_0} &= 0 + 1 \\
R_{w_0,w_1} &= 1 \\
R_{w_1,w_2} &= 0 \\
R_{w_1,w_1} &= \varnothing \quad \text{(no self-loop on } w_1\text{)}
\end{aligned}$$

2. Step 2: Eliminate State w_1

   We eliminate the intermediate state w_1 using the formula:

$$R_{ij}^{\text{new}} = R_{ij}^{\text{old}} + R_{ik}(R_{kk})^* R_{kj}$$

   For the path from w_0 to w_2 through w_1:

$$\begin{aligned}
R_{w_0,w_2}^{\text{new}} &= R_{w_0,w_2}^{\text{old}} + R_{w_0,w_1}(R_{w_1,w_1})^* R_{w_1,w_2} \\
&= \varnothing + 1 \cdot (\varnothing)^* \cdot 0 \\
&= 1 \cdot \varepsilon \cdot 0 \\
&= 10
\end{aligned}$$

   After eliminating w_1, we have:

$$\begin{aligned}
R_{w_0,w_0} &= 0 + 1 \\
R_{w_0,w_2} &= 10
\end{aligned}$$

3. Step 3: Construct Final Regular Expression

   Starting from w_0 and reaching the accepting state w_2. We can loop at w_0 any number of times using $(0 + 1)^*$ Then take the direct path to w_2 using 10

   The final regular expression is:

$$\boxed{(0 + 1)^* 10}$$

4. Step 4: Verification

   Examples of accepted strings:

(a) 10: $w_0 \xrightarrow{1} w_0, w_1 \xrightarrow{0} w_2$

(b) 010: $w_0 \xrightarrow{0} w_0 \xrightarrow{1} w_0, w_1 \xrightarrow{0} w_2$

(c) 11110: $w_0 \xrightarrow{1} w_0, w_1 \xrightarrow{1} w_0 \xrightarrow{1} w_0, w_1 \xrightarrow{1} w_0 \xrightarrow{1} w_0, w_1 \xrightarrow{0} w_2$

The language is: *all strings over* $\{0, 1\}$ *that end with* 10.

7.4 DFA to RegExp

To convert a DFA to a regular expression, we can use state elimination as well. The process is almost the same as for NFAs:

1. Represent each transition as a regular expression.
2. Identify the start and accepting states.
3. Iteratively eliminate intermediate states using:

$$R_{ij}^{\text{new}} = R_{ij}^{\text{old}} + R_{ik}(R_{kk})^* R_{kj}$$

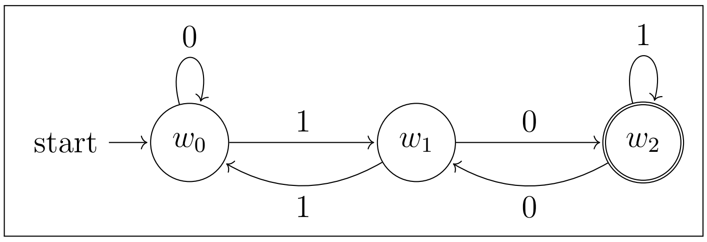
4. After eliminating all intermediate states, the resulting regex from the start to accepting state is the regular expression of the language.

Example:
Let the DFA be:

$$W = \{w_0, w_1, w_2\}, \quad \Sigma = \{0, 1\}, \quad S = \{w_0\}, \quad F = \{w_2\}$$

Transition function V:

$$\begin{aligned} V(w_0, 0) = w_0, \quad V(w_0, 1) = w_1 \\ V(w_1, 0) = w_2, \quad V(w_1, 1) = w_0 \\ V(w_2, 0) = w_1, \quad V(w_2, 1) = w_2 \end{aligned}$$



1. Step 1:

   We label each transition with its corresponding regular expression:

$$w_0 \xrightarrow{0} w_0, \quad w_0 \xrightarrow{1} w_1, \quad w_1 \xrightarrow{0} w_2, \quad w_1 \xrightarrow{1} w_0, \quad w_2 \xrightarrow{0} w_1, \quad w_2 \xrightarrow{1} w_2$$

2. Step 2:

   We eliminate the intermediate state w_1 using the formula:

   $$R_{ij}^{\text{new}} = R_{ij}^{\text{old}} + R_{ik}(R_{kk})^* R_{kj}$$

   First, we need to find R_{w_1,w_1}. Although w_1 has no direct self-loop, it can return to itself via w_0:

   $$w_1 \xrightarrow{1} w_0 \xrightarrow{0^*} w_0 \xrightarrow{1} w_1$$

   Therefore:

   $$R_{w_1,w_1} = 10^*1$$

   Path from w_0 to w_2 through w_1:

   $$\begin{aligned} R_{w_0,w_2}^{\text{new}} &= R_{w_0,w_2}^{\text{old}} + R_{w_0,w_1}(R_{w_1,w_1})^* R_{w_1,w_2} \\ &= \varnothing + 1 \cdot (10^*1)^* \cdot 0 \\ &= 1(10^*1)^*0 \end{aligned}$$

   Self-loop on w_0:

   $$\begin{aligned} R_{w_0,w_0}^{\text{new}} &= R_{w_0,w_0}^{\text{old}} + R_{w_0,w_1}(R_{w_1,w_1})^* R_{w_1,w_0} \\ &= 0 + 1 \cdot (10^*1)^* \cdot 1 \\ &= 0 + 1(10^*1)^*1 \end{aligned}$$

   Path from w_2 to w_2:

   $$\begin{aligned} R_{w_2,w_2}^{\text{new}} &= R_{w_2,w_2}^{\text{old}} + R_{w_2,w_1}(R_{w_1,w_1})^* R_{w_1,w_2} \\ &= 1 + 0 \cdot (10^*1)^* \cdot 0 \\ &= 1 + 0(10^*1)^*0 \end{aligned}$$

   Then, the path from w_0 to w_2 is:

   $$R_{w_0,w_2} = 1(10^*1)^*0$$

   We can loop at w_0 using $(0 + 1(10^*1)^*1)^*$, then go to w_2 and loop there using $(1 + 0(10^*1)^*0)^*$. The final regular expression for strings accepted by this DFA is:

   $$\boxed{R = (0 + 1(10^*1)^*1)^* \cdot 1(10^*1)^*0 \cdot (1 + 0(10^*1)^*0)^*}$$

3. Step 3:

   Examples of accepted strings:

   (a) 10: $w_0 \xrightarrow{1} w_1 \xrightarrow{0} w_2$

(b) 110: $w_0 \xrightarrow{1} w_1 \xrightarrow{1} w_0 \xrightarrow{1} w_1 \xrightarrow{0} w_2$

(c) 101: $w_0 \xrightarrow{1} w_1 \xrightarrow{0} w_2 \xrightarrow{1} w_2$

(d) 0101: $w_0 \xrightarrow{0} w_0 \xrightarrow{1} w_1 \xrightarrow{0} w_2 \xrightarrow{1} w_2$

The language is: *all strings over* $\{0,1\}$ *that contain* 10 *as a substring.*

7.5 RegExp to Finite Automaton FA

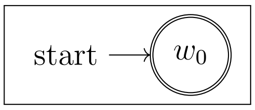
To convert a regular expression to a finite automaton (FA), we can use the following construction rules:

1. For the empty string ε, create an FA with a single state that is both the start and accepting state.

   Example:

   The regular expression ε is accepted by an FA consisting of one state w_0, where:

   $$S = \{w_0\}, \quad F = \{w_0\}$$

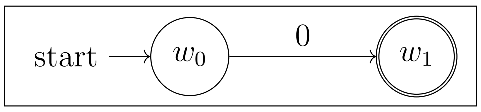
   No transitions are required. The empty string is accepted immediately.

2. For a single symbol $a \in \Sigma$, create an FA with two states: a start state and an accepting state, with a transition labeled a from the start to the accepting state.

   Example:

   For the regular expression 0, construct an FA with states w_0 and w_1 such that:

   $$S = \{w_0\}, \quad F = \{w_1\}, \quad V(w_0, 0) = \{w_1\}$$

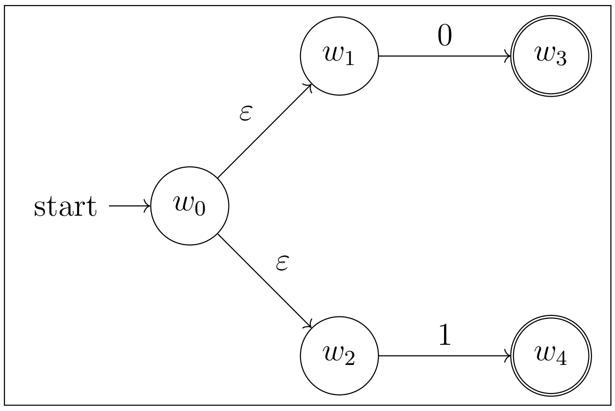
   

   The string 0 is accepted, while all other strings are rejected.

3. For union $R_1 + R_2$, create a new start state with ε-transitions to the start states of the FAs for R_1 and R_2. The accepting states of both FAs become accepting states of the new FA.

   Example:

   Let $R_1 = 0$ and $R_2 = 1$.

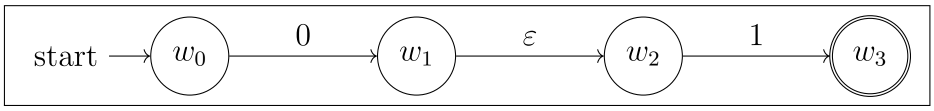


The resulting FA accepts either 0 or 1.

4. For concatenation R_1R_2, connect the accepting states of the FA for R_1 to the start state of the FA for R_2 using ε-transitions.

   Example:

   Let $R_1 = 0$ and $R_2 = 1$.

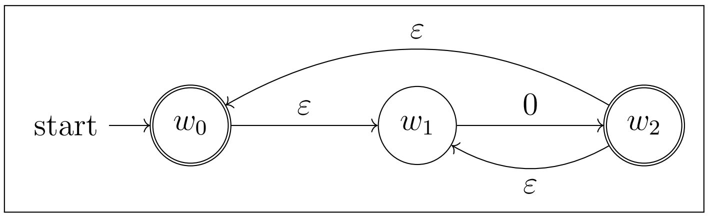


   The resulting FA accepts the string 01.

5. For Kleene star R^*, create a new start state that is also an accepting state. Add ε-transitions to allow repetition.

   Example:

   Let $R = 0$.



   This FA accepts $\varepsilon, 0, 00, 000$, and so on.

7.6 Moore to Mealy

To convert a Moore machine to an equivalent Mealy machine, we adjust the output function so that outputs are associated with transitions rather than states.

Given a Moore machine defined as:

$$M_o = (W, \Sigma, \Delta, V, \lambda, S),$$

we construct an equivalent Mealy machine:

$$M_e = (W, \Sigma, \Delta, V, \lambda', S),$$

where the new output function λ' is defined by:

$$\lambda'(w, x) = \lambda(V(w, x)) \quad \forall\, w \in W,\ \forall\, x \in \Sigma.$$

This ensures that the Mealy machine produces, during each transition, the same output that the Moore machine produces upon entering the next state.

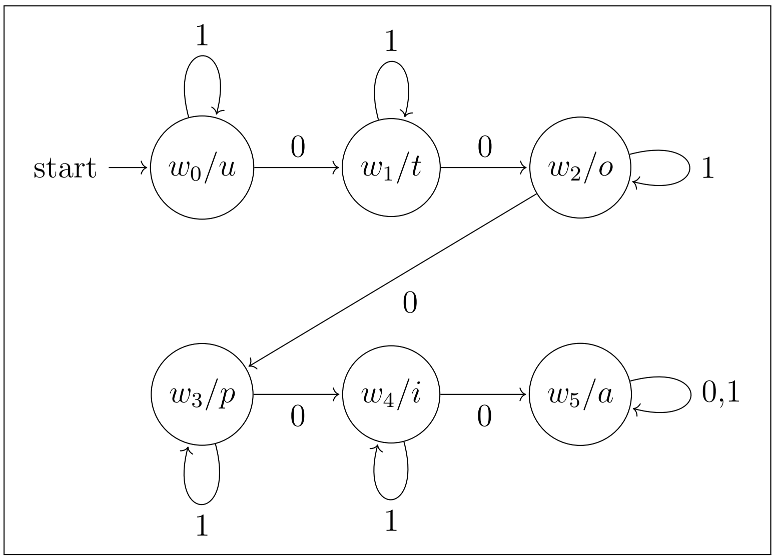
Example:

Let $W = \{w_0, w_1, w_2, w_3, w_4, w_5\}$, $\Sigma = \{0, 1\}$, $\Delta = \{u, t, o, p, i, a\}$, and $S = w_0$.

1. **Moore Machine**

   Transition Function V and Output Function λ:

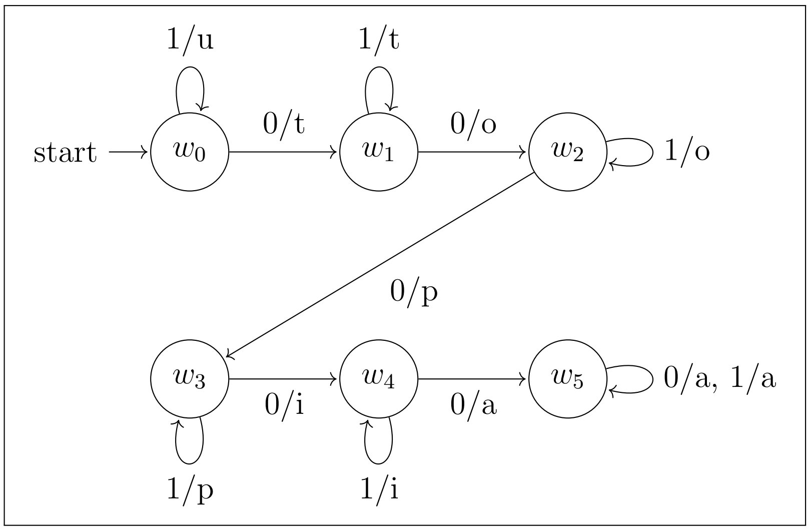
W	**0**	**1**	**Output**
w_0	w_1	w_0	u
w_1	w_2	w_1	t
w_2	w_3	w_2	o
w_3	w_4	w_3	p
w_4	w_5	w_4	i
w_5	w_5	w_5	a



2. **Mealy Machine**

   Using $\lambda'(w, x) = \lambda(V(w, x))$, we compute:

W	**0**	**1**
w_0	w_1/t	w_0/u
w_1	w_2/o	w_1/t
w_2	w_3/p	w_2/o
w_3	w_4/i	w_3/p
w_4	w_5/a	w_4/i
w_5	w_5/a	w_5/a



3. **Comparison on input string** 00000:

Input	**State**	**Moore Output**	**Mealy Output**
–	w_0	u	–
0	w_1	t	t
0	w_2	o	o
0	w_3	p	p
0	w_4	i	i
0	w_5	a	a

Hence, both machines produce the same output sequence: $\boxed{utopia}$

7.7 Mealy to Moore

To convert a Mealy machine to an equivalent Moore machine, we must modify the state space so that outputs are associated with states rather than transitions.

Let the Mealy machine be defined as

$$M_e = (W, \Sigma, \Delta, V, \lambda, S),$$

where $\lambda : W \times \Sigma \to \Delta$.

We construct an equivalent Moore machine

$$M_o = (W', \Sigma, \Delta, V', \lambda', S')$$

as follows.

1. State set

$$W' = \{\, (w, y) \mid w \in W,\ y \in \Delta,\ \exists x \in \Sigma \text{ such that } \lambda(w, x) = y \,\}$$

Each state (w, y) represents being in Mealy state w with output y.

2. Initial state

   Since a Mealy machine produces output only after reading an input symbol, we introduce a new symbol $s_\perp \notin W'$ to serve as the initial state.

$$S' = s_\perp$$

3. Output function

$$\lambda'(s_\perp) = \varepsilon$$
$$\lambda'((w, y)) = y \quad \forall (w, y) \in W'$$

4. Transition function

   From the initial symbol:

$$V'(s_\perp, x) = (V(S, x), \lambda(S, x)) \quad \forall x \in \Sigma$$

   For all other states:

$$V'((w, y), x) = (V(w, x), \lambda(w, x)) \quad \forall (w, y) \in W',\ \forall x \in \Sigma$$

The resulting Moore machine is equivalent to the original Mealy machine with respect to input/output behavior. The construction may introduce unreachable states; therefore, only states reachable from S' need to be retained to obtain a minimal equivalent Moore machine.

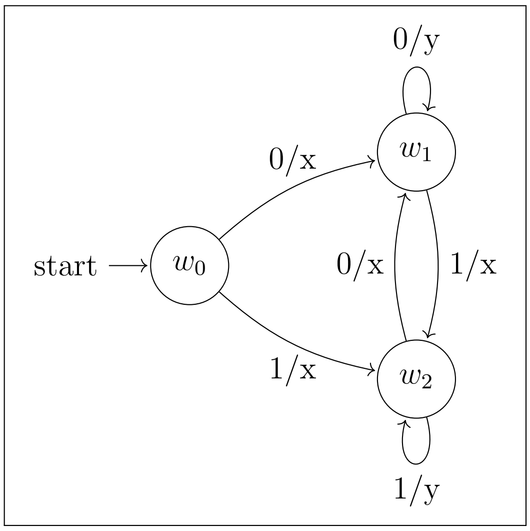
Example:

1. **Mealy Machine**

$$W = \{w_0, w_1, w_2\}, \quad \Sigma = \{0, 1\}, \quad \Delta = \{x, y\}, \quad S = w_0$$

   With this transition

W	0	1
w_0	w_1/x	w_2/x
w_1	w_1/y	w_2/x
w_2	w_1/x	w_2/y



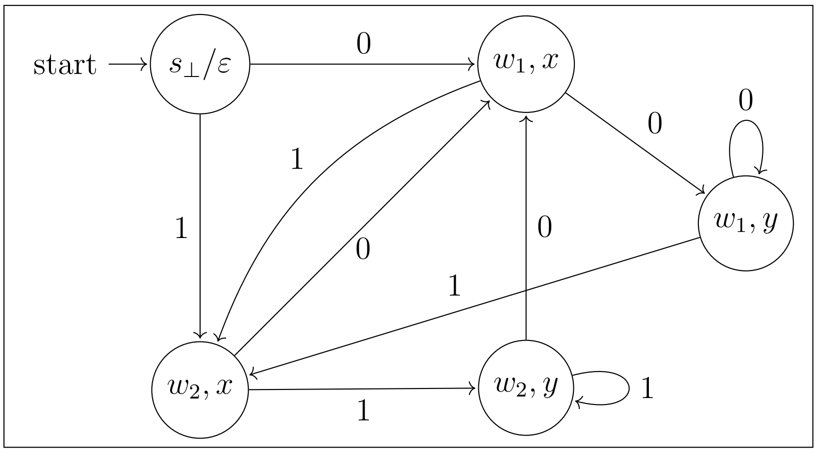
2. **Moore Machine**

   All reachable states:

$$W' = \{s_\perp \cup (w_1, x), (w_1, y), (w_2, x), (w_2, y)\}$$

   With this transition

W'	0	1
$s_\perp$	w_1, x	w_2, x
w_1, x	w_1, y	w_2, x
w_1, y	w_1, y	w_2, x
w_2, x	w_1, x	w_2, y
w_2, y	w_1, x	w_2, y



8 Property of Regular Languages

8.1 Decision Algorithms

A language is regular if there exists a finite automaton that recognizes it. The following decision problems can be solved for regular languages:

8.1.1 Emptiness

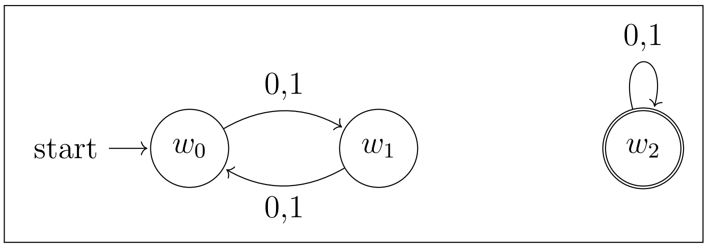
$$L(M) = \varnothing \leftrightarrow \forall f \in F,\ f \text{ is not reachable from } S$$

A DFA M has an **empty language** if no accepting state can be reached from the initial state.

Example:

Let $M = (W, \Sigma, V, S, F)$ where

$$W = \{w_0, w_1, w_2\}, \quad \Sigma = \{0, 1\}, \quad S = w_0, \quad F = \{w_2\}.$$



The accepting state w_2 is isolated and unreachable from the initial state w_0. Since there is no path from w_0 to w_2, no string can be accepted. Hence:

$$L(M) = \varnothing.$$

8.1.2 Finiteness

A language $L(M)$ is finite if it contains a finite number of strings, and infinite otherwise.

1. **Finite Language**

$$L(M) \text{ is finite} \leftrightarrow \nexists w' \in W : S \xrightarrow{*} w' \xrightarrow{+} w' \xrightarrow{*} f,\ f \in F$$

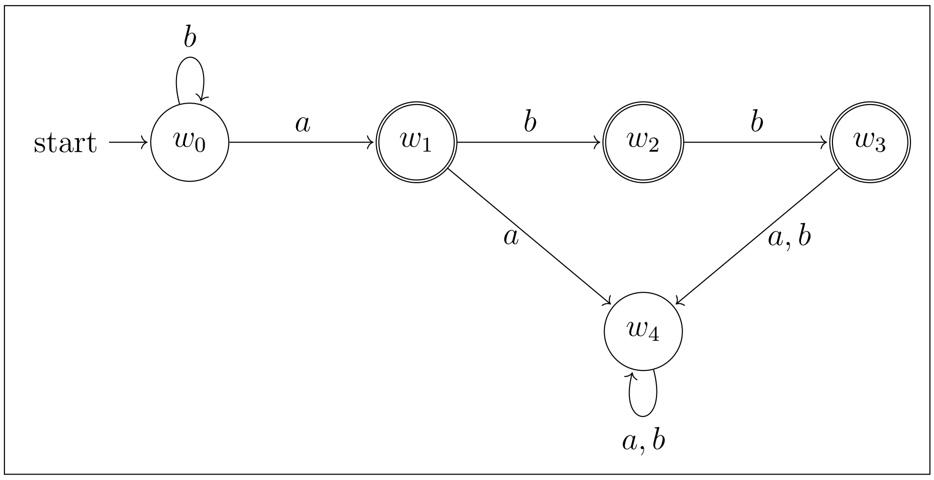
A language is finite if there is no cycle on any path from the initial state to an accepting state.

Example:

DFA M_1 accepts $L = \{a, ab, abb\}$. Since $|L| = 3 < \infty$, the language is finite.

Let $M_1 = (W, \Sigma, V, S, F)$ where:

$$W = \{w_0, w_1, w_2, w_3, w_4\}, \quad \Sigma = \{a, b\}, \quad S = w_0, \quad F = \{w_1, w_2, w_3\}.$$



The language is finite because there are no cycles on any path from S to accepting states $\{w_1, w_2, w_3\}$.

2. **Infinite Language**

$$L(M) \text{ is infinite} \leftrightarrow \exists w' \in W : S \xrightarrow{*} w' \xrightarrow{+} w' \xrightarrow{*} f,\ f \in F$$

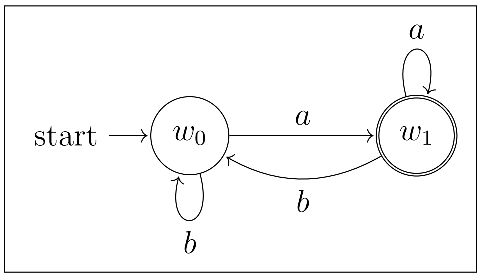
A language is infinite if there exists a cycle on some path from the initial state to an accepting state.

Example:

DFA M_2 accepts $L = \{a^n \mid n \geq 1\}$, which is infinite.

Let $M_2 = (W, \Sigma, V, S, F)$ where:

$$W = \{w_0, w_1\}, \quad \Sigma = \{a, b\}, \quad S = w_0, \quad F = \{w_1\}.$$



The language is infinite because state w_1 has a self-loop (cycle), is reachable from S, and is an accepting state, satisfying

$$S \xrightarrow{*} w_1 \xrightarrow{+} w_1.$$

8.1.3 Membership

$$x \in L(M) \leftrightarrow \hat{V}(S, x) \in F$$

where $\hat{V}$ is the extended transition function that processes the entire string x from the start state S.

Example:

Given DFA $M = (W, \Sigma, V, S, F)$ with

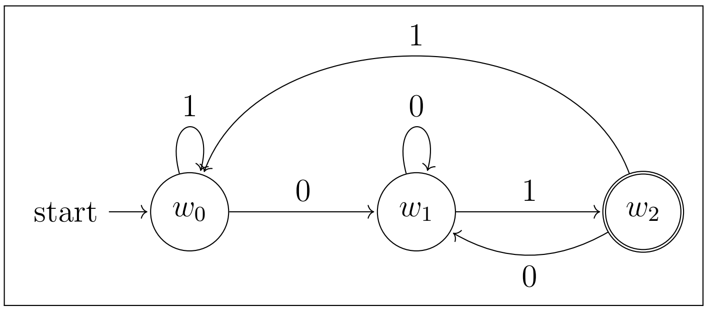
$$L(M) = \{w \in \{0,1\}^* \mid w \text{ ends in } 01\}$$

where:

$$W = \{w_0, w_1, w_2\}, \quad \Sigma = \{0, 1\}, \quad S = w_0, \quad F = \{w_2\}.$$

Transition table:

V	0	1
w_0	w_1	w_0
w_1	w_1	w_2
w_2	w_1	w_0



Testing membership:

$$
\begin{aligned}
V(w_0, 1) &= w_0 \\
V(w_0, 0) &= w_1 \\
V(w_1, 0) &= w_1 \\
V(w_1, 1) &= w_2
\end{aligned}
$$

Since $V^*(w_0, 1001) = w_2 \in F$, the string $x = 1001$ is accepted. Therefore,

$$x \in L(M).$$

The string 1001 ends in 01, confirming membership in the language.

8.1.4 Equivalence

$$M_1 \equiv M_2 \leftrightarrow L(M_1) = L(M_2)$$

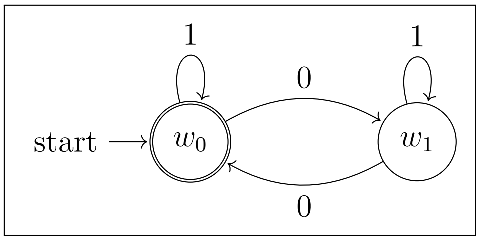
Two DFAs M_1 and M_2 are equivalent if and only if they accept exactly the same language, i.e., for all strings $x \in \Sigma^*$:

$$x \in L(M_1) \leftrightarrow x \in L(M_2)$$

Example:
Consider two DFAs that both accept strings with an even number of 0's over $\Sigma = \{0, 1\}$.
DFA M_1:
$M_1 = (W_1, \Sigma, V_1, S_1, F_1)$ where:

$$W_1 = \{w_0, w_1\}, \quad \Sigma = \{0, 1\}, \quad S_1 = w_0, \quad F_1 = \{w_0\}.$$

State	0	1
w_0	w_1	w_0
w_1	w_0	w_1



DFA M_2:
$M_2 = (W_2, \Sigma, V_2, S_2, F_2)$ where:

$$W_2 = \{v_0, v_1, v_2, v_3\}, \quad \Sigma = \{0, 1\}, \quad S_2 = v_0, \quad F_2 = \{v_0, v_2\}.$$

State	0	1
v_0	v_1	v_2
v_1	v_0	v_3
v_2	v_3	v_0
v_3	v_2	v_1

$$\begin{tikzcd}
\text{start} \arrow[r] & v_0 \arrow[r, bend left, "0"] \arrow[d, bend left, "1"] & v_1 \arrow[l, bend left, "0"] \arrow[d, bend left, "1"] \\
& v_2 \arrow[r, bend left, "0"] \arrow[u, bend left, "1"] & v_3 \arrow[l, bend left, "0"] \arrow[u, bend left, "1"]
\end{tikzcd}$$

Let's test several strings to verify both DFAs accept the same language:

1. $x = \varepsilon$: M_1 ends at w_0, M_2 ends at v_0 (0 zeros, even)
2. $x = 00$: M_1 ends at w_0, M_2 ends at v_0 (2 zeros, even)
3. $x = 101$: M_1 ends at w_1, M_2 ends at v_3 (1 zero, odd)
4. $x = 0110$: M_1 ends at w_0, M_2 ends at v_2 (2 zeros, even)

Both DFAs accept exactly the same language:

$$L(M_1) = L(M_2) = \{w \in \{0,1\}^* \mid \text{number of 0's in } w \text{ is even}\}$$

Despite having different numbers of states and different structures, they recognize the same language. Therefore, $M_1 \equiv M_2$.

M_2 tracks both the parity of 0's (even/odd) and whether the last symbol was 0 or 1, making it non-minimal. States v_0 and v_2 are both accepting because they both represent an even number of 0's.

8.1.5 Minimization

A minimal DFA M' for a language L is a DFA such that

$$L(M') = L \ \wedge \ \nexists \text{ DFA } M'' \text{ with } L(M'') = L \text{ and } |W''| < |W'|.$$

For any DFA M with $L(M) = L$, the minimization process produces such an automaton M' by merging equivalent states, where $w \sim w'$ denotes that the states w and w' are equivalent.

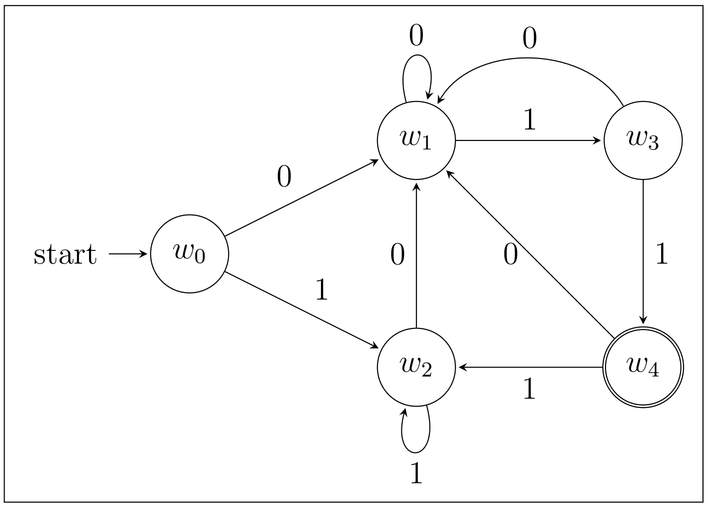
Example 1:

Let $M = (W, \Sigma, V, S, F)$, where

$$W = \{w_0, w_1, w_2, w_3, w_4\}, \quad \Sigma = \{0,1\}, \quad S = w_0, \quad F = \{w_4\}.$$

The transition function is given by:

W	0	1
w_0	w_1	w_2
w_1	w_1	w_3
w_2	w_1	w_2
w_3	w_1	w_4
w_4	w_1	w_2



1. For a DFA M, the n-equivalence relation $\equiv_n$ on states is defined inductively as follows:
$$w \equiv_0 w' \;\leftrightarrow\; (w \in F \leftrightarrow w' \in F).$$
$$w \equiv_{n+1} w' \;\leftrightarrow\; \big(w \equiv_0 w' \;\wedge\; \forall a \in \Sigma,\; V(w,a) \equiv_n V(w',a)\big).$$

2. The equivalence relation used in DFA minimization is obtained when this sequence stabilizes:
$$\exists k \in \mathbb{N} \text{ such that } \equiv_k = \equiv_{k+1} .$$

3. The resulting equivalence relation can be characterized as:
$$w \sim w' \;\leftrightarrow\; \forall x \in \Sigma^*,\; \big(V^*(w,x) \in F \leftrightarrow V^*(w',x) \in F\big).$$

From the transition structure, we compute the successive partitions:

1. **0-Equivalence:** Partition states by acceptance.
$$\phi_0 = \big\{\{w_0, w_1, w_2, w_3\}, \{w_4\}\big\}$$

2. **1-Equivalence:** Refine partitions based on transitions.

   For $\{w_0, w_1, w_2, w_3\}$:

   (a) w_0: $0 \to w_1$, $1 \to w_2$

   (b) w_1: $0 \to w_1$, $1 \to w_3$

   (c) w_2: $0 \to w_1$, $1 \to w_2$

   (d) w_3: $0 \to w_1$, $1 \to w_4$

   Since w_3 transitions to an accepting state on input 1, we split:
$$\phi_1 = \big\{\{w_0, w_1, w_2\}, \{w_3\}, \{w_4\}\big\}$$

3. **2-Equivalence:** Refine $\{w_0, w_1, w_2\}$.

   (a) w_0: $1 \to w_2 \in \{w_0, w_2\}$

(b) w_1: $1 \to w_3 \in \{w_3\}$

(c) w_2: $1 \to w_2 \in \{w_0, w_2\}$

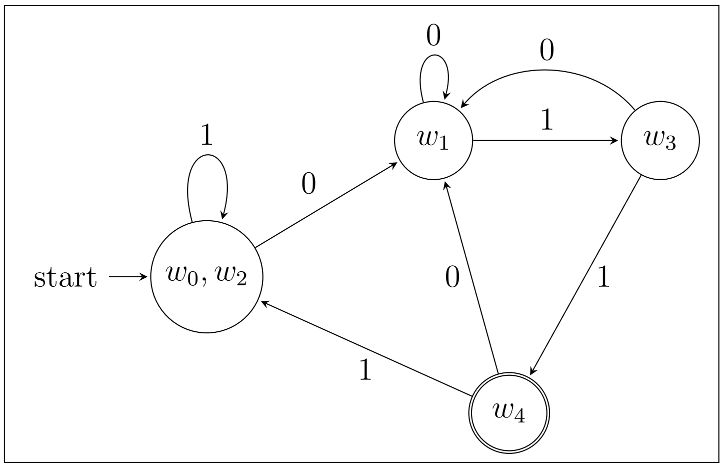
Thus,

$$\phi_2 = \big\{\{w_0, w_2\}, \{w_1\}, \{w_3\}, \{w_4\}\big\}$$

4. **3-Equivalence:** The block $\{w_0, w_2\}$ cannot be further refined, so

$$\phi_3 = \phi_2.$$

Hence, the minimal DFA merges w_0 and w_2 into a single state. The minimized DFA has states $\{\{w_0, w_2\}, w_1, w_3, w_4\}$.



Example 2:
Let

$$M = (W, \Sigma, V, S, F)$$

where

$$W = \{w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7\}, \quad \Sigma = \{0, 1\}, \quad S = w_0, \quad F = \{w_2\}.$$

The transition function is:

W	0	1
w_0	w_1	w_5
w_1	w_6	w_2
w_2	w_0	w_2
w_3	w_2	w_6
w_4	w_7	w_5
w_5	w_2	w_6
w_6	w_6	w_4
w_7	w_6	w_2

1. **0-Equivalence**

   Partition states by acceptance:

$$f_0 = \big\{\{w_2\}, \{w_0, w_1, w_3, w_4, w_5, w_6, w_7\}\big\}$$

2. **1-Equivalence**

   Refine the non-accepting block by transitions:

W	0	1
w_0	w_1	w_5
w_1	w_6	w_2
w_3	w_2	w_6
w_4	w_7	w_5
w_5	w_2	w_6
w_6	w_6	w_4
w_7	w_6	w_2

   States transitioning to the accepting state w_2 are separated:

$$f_1 = \big\{\{w_2\}, \{w_1, w_3, w_5, w_7\}, \{w_0, w_4, w_6\}\big\}$$

3. **2-Equivalence**

   Refine each block further. For $\{w_1, w_3, w_5, w_7\}$:

W	0	1
w_1	w_6	w_2
w_3	w_2	w_6
w_5	w_2	w_6
w_7	w_6	w_2

   This splits into:

$$\{w_1, w_7\}, \quad \{w_3, w_5\}$$

   For $\{w_0, w_4, w_6\}$:

W	0	1
w_0	w_1	w_5
w_4	w_7	w_5
w_6	w_6	w_4

   This splits into:

$$\{w_0, w_4\}, \quad \{w_6\}$$

   Thus:

$$f_2 = \big\{\{w_2\}, \{w_1, w_7\}, \{w_3, w_5\}, \{w_0, w_4\}, \{w_6\}\big\}$$

4. **3-Equivalence**

   All blocks are stable under further refinement:

$$f_3 = f_2$$

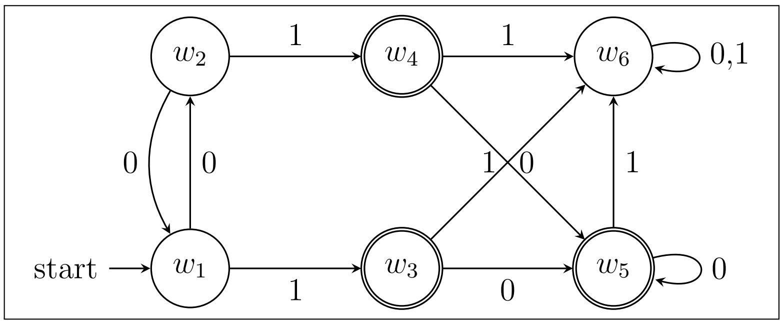
The equivalence classes of the minimized DFA are:

$$\{w_2\},\ \{w_1, w_7\},\ \{w_3, w_5\},\ \{w_0, w_4\},\ \{w_6\}$$

Hence, the minimal DFA has 5 states

8.1.6 Myhill-Nerode Theorem

We can also minimize DFA with Myhill-Nerode Theorem by following these steps
Example:
Suppose we have a DFA M as below:



1. Step 1: Draw a table of all pair of statess (X,Y).

	w_1	w_2	w_3	w_4	w_5	w_6
w_1	–		✓	✓	✓	
w_2		–	✓	✓	✓	
w_3			–			✓
w_4				–		✓
w_5					–	✓
w_6						–

2. step 2: Mark all pairs where $(X \in F \wedge Y \notin F) \vee (X \notin F \wedge Y \in F)$

$$(w_1, w_2) \begin{cases} V(w_1, 0) = w_2, & V(w_2, 0) = w_1, \\ V(w_1, 1) = w_3, & V(w_2, 1) = w_4 \end{cases}$$

$$(w_3, w_4) \begin{cases} V(w_3, 0) = w_5, & V(w_4, 0) = w_5, \\ V(w_3, 1) = w_6, & V(w_4, 1) = w_6 \end{cases}$$

$$(w_3, w_5) \begin{cases} V(w_3, 0) = w_5, & V(w_5, 0) = w_5, \\ V(w_3, 1) = w_6, & V(w_5, 1) = w_6 \end{cases}$$

$$(w_4, w_5) \begin{cases} V(w_4, 0) = w_5, & V(w_5, 0) = w_5, \\ V(w_4, 1) = w_6, & V(w_5, 1) = w_6 \end{cases}$$

$$(w_1, w_6) \begin{cases} V(w_1, 0) = w_2, & V(w_6, 0) = w_6, \\ V(w_1, 1) = w_3, & V(w_6, 1) = w_6 \end{cases}$$

$$(w_2, w_6) \begin{cases} V(w_2, 0) = w_1, & V(w_6, 0) = w_6, \\ V(w_2, 1) = w_4, & V(w_6, 1) = w_6 \end{cases}$$

3. step 3: Iteratively mark pairs where transitions lead to already marked pairs until no new pairs can be marked.

$$(w_1, w_2) \begin{cases} V(w_1, 0) = w_2, & V(w_2, 0) = w_1, \\ V(w_1, 1) = w_3, & V(w_2, 1) = w_4 \end{cases}$$

$$(w_3, w_4) \begin{cases} V(w_3, 0) = w_5, & V(w_4, 0) = w_5, \\ V(w_3, 1) = w_6, & V(w_4, 1) = w_6 \end{cases}$$

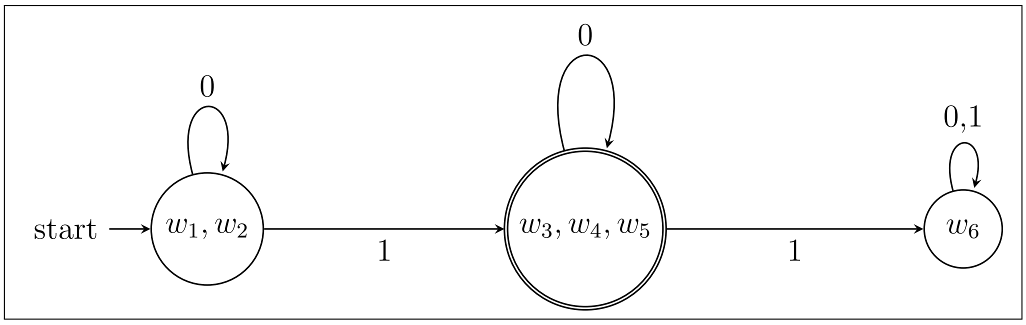
$$(w_3, w_5) \begin{cases} V(w_3, 0) = w_5, & V(w_5, 0) = w_5, \\ V(w_3, 1) = w_6, & V(w_5, 1) = w_6 \end{cases}$$

$$(w_4, w_5) \begin{cases} V(w_4, 0) = w_5, & V(w_5, 0) = w_5, \\ V(w_4, 1) = w_6, & V(w_5, 1) = w_6 \end{cases}$$

4. step 4: Combine all the unmarked pairs into equivalence classes.

$$\{w_1, w_2\},\ \{w_3, w_4, w_5\},\ \{w_6\}$$

Hence the final diagram is:



8.1.7 Subset

$$L(M_1) \subseteq L(M_2) \;\leftrightarrow\; \forall x \in \Sigma^*,\; x \in L(M_1) \to x \in L(M_2)$$

Example:
Let

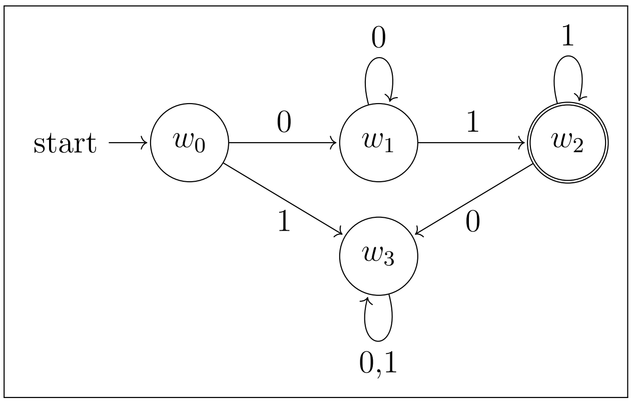
$$L(M_1) = \{0^n 1^m \mid n \geq 1,\; m \geq 1\}, \qquad L(M_2) = \{0^n 1^m \mid n \geq 0,\; m \geq 0\}.$$

DFA M_1:

$$M_1 = (W_1, \Sigma, V_1, S_1, F_1)$$

where

$$W_1 = \{w_0, w_1, w_2, w_3\}, \quad \Sigma = \{0, 1\}, \quad S_1 = w_0, \quad F_1 = \{w_2\}.$$



This DFA accepts exactly the language

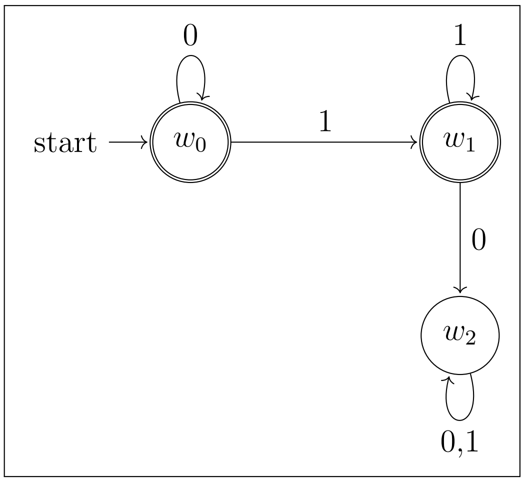
$$L(M_1) = \{0^n 1^m \mid n \geq 1,\; m \geq 1\}.$$

DFA M_2:

$$M_2 = (W_2, \Sigma, V_2, S_2, F_2)$$

where

$$W_2 = \{w_0, w_1, w_2\}, \quad \Sigma = \{0, 1\}, \quad S_2 = w_0, \quad F_2 = \{w_0, w_1\}.$$



This DFA accepts exactly

$$L(M_2) = \{0^n 1^m \mid n \geq 0,\ m \geq 0\} = 0^*1^*.$$

Since every string with at least one 0 followed by at least one 1 is also a string of the form 0^*1^*, we conclude that

$$L(M_1) \subseteq L(M_2).$$

8.2 Closure Properties

8.2.1 Union

$$L(M_1) \cup L(M_2) = \{x \in \Sigma^* \mid x \in L(M_1) \lor x \in L(M_2)\}$$

Given $M_1 = (W_1, \Sigma, V_1, S_1, F_1)$ and $M_2 = (W_2, \Sigma, V_2, S_2, F_2)$, the union is constructed as:

$$M = (W_1 \times W_2, \Sigma, V, (S_1, S_2), F)$$

where

$$F = \{(w_1, w_2) \mid w_1 \in F_1 \lor w_2 \in F_2\}$$

and

$$V((w_1, w_2), a) = (V_1(w_1, a), V_2(w_2, a))$$

Example:

Let $L(M_1)$ be the set of strings over $\{a, b\}$ containing at least one a, and let $L(M_2)$ be the set of strings containing at least one b.

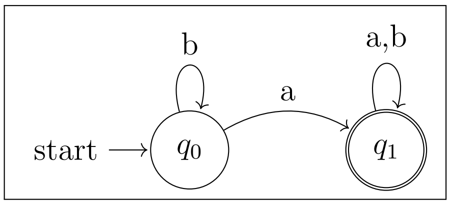
The union language is:

$$L(M_1) \cup L(M_2) = \{w \in \{a, b\}^* \mid w \text{ contains at least one } a \text{ or one } b\}.$$

1. Component DFA M_1: Accepts strings with at least one a.

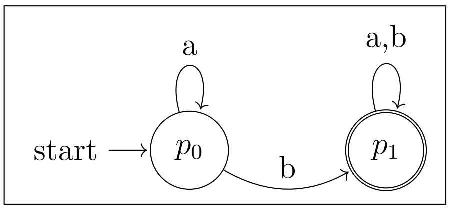
   $M_1 = (W_1, \Sigma, V_1, S_1, F_1)$ where

$$W_1 = \{q_0, q_1\}, \quad \Sigma = \{a, b\}, \quad S_1 = q_0, \quad F_1 = \{q_1\}.$$



2. Component DFA M_2: Accepts strings with at least one b.
   $M_2 = (W_2, \Sigma, V_2, S_2, F_2)$ where

$$W_2 = \{p_0, p_1\}, \quad \Sigma = \{a, b\}, \quad S_2 = p_0, \quad F_2 = \{p_1\}.$$



Union Construction:

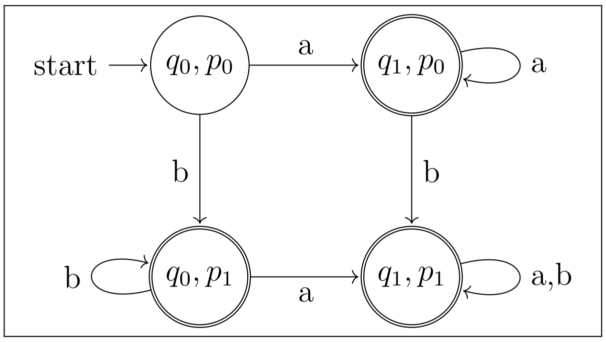
$$M = M_1 \cup M_2 = (W_1 \times W_2,\ \Sigma,\ V,\ (q_0, p_0),\ F)$$

where

$$W = \{(q_0, p_0), (q_0, p_1), (q_1, p_0), (q_1, p_1)\}$$

and

$$F = \{(x, y) \mid x \in F_1 \vee y \in F_2\} = \{(q_1, p_0), (q_0, p_1), (q_1, p_1)\}.$$



The product state (q_i, p_j) represents "M_1 in state q_i and M_2 in state p_j". A state is accepting if *either* component is in an accepting state (union semantics).

Thus,

$$L(M_1) \cup L(M_2) = L(M) = \{w \in \{a, b\}^* \mid w \neq \varepsilon\}.$$

Verification:

(a) ε: Neither M_1 nor M_2 accepts $\rightarrow$ rejected
(b) a: M_1 accepts $\rightarrow$ accepted
(c) b: M_2 accepts $\rightarrow$ accepted
(d) ab: Both accept $\rightarrow$ accepted

8.2.2 Intersection

$$L(M_1) \cap L(M_2) = \{x \in \Sigma^* \mid x \in L(M_1) \wedge x \in L(M_2)\}$$

Given $M_1 = (W_1, \Sigma, V_1, S_1, F_1)$ and $M_2 = (W_2, \Sigma, V_2, S_2, F_2)$, the intersection is constructed as:

$$M = (W_1 \times W_2, \Sigma, V, (S_1, S_2), F)$$

where

$$F = \{(w_1, w_2) \mid w_1 \in F_1 \wedge w_2 \in F_2\}$$

and

$$V((w_1, w_2), a) = (V_1(w_1, a), V_2(w_2, a))$$

Example:

Let $L(M_1)$ be the set of strings over $\{a, b\}$ containing at least one a, and let $L(M_2)$ be the set of strings containing at least one b.

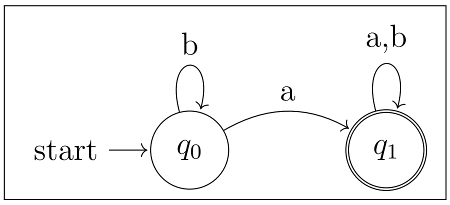
The intersection language is:

$$L(M_1) \cap L(M_2) = \{w \in \{a, b\}^* \mid w \text{ contains at least one } a \text{ and at least one } b\}.$$

1. Component DFA M_1: Accepts strings with at least one a.

   $M_1 = (W_1, \Sigma, V_1, S_1, F_1)$ where

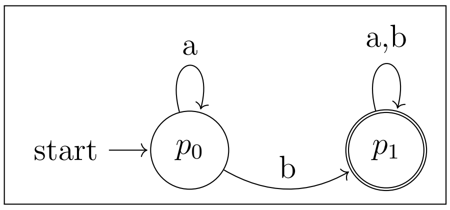
   $$W_1 = \{q_0, q_1\}, \quad \Sigma = \{a, b\}, \quad S_1 = q_0, \quad F_1 = \{q_1\}.$$

2. Component DFA M_2: Accepts strings with at least one b.

   $M_2 = (W_2, \Sigma, V_2, S_2, F_2)$ where

   $$W_2 = \{p_0, p_1\}, \quad \Sigma = \{a, b\}, \quad S_2 = p_0, \quad F_2 = \{p_1\}.$$

   Intersection construction:

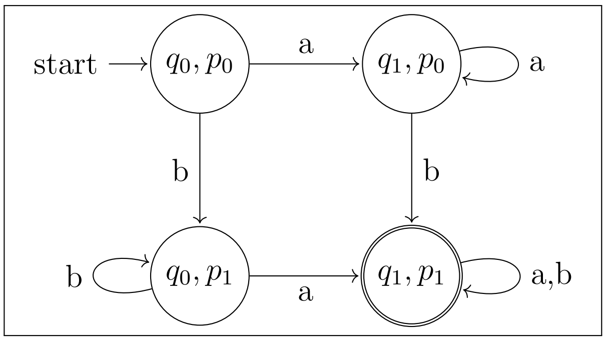
   $$M = M_1 \cap M_2 = (W_1 \times W_2,\ \Sigma,\ V,\ (q_0, p_0),\ F)$$

   where

   $$W = \{(q_0, p_0), (q_0, p_1), (q_1, p_0), (q_1, p_1)\}$$

   and

   $$F = \{(x, y) \mid x \in F_1 \wedge y \in F_2\} = \{(q_1, p_1)\}.$$



The product state (q_i, p_j) represents "M_1 in state q_i and M_2 in state p_j". A state is accepting if *both* components are in accepting states (intersection semantics).

Thus,

$$L(M_1) \cap L(M_2) = L(M) = \{w \in \{a, b\}^* \mid w \text{ contains both } a \text{ and } b\}.$$

Verification:

(a) ε: Neither contains a nor $b \to$ rejected
(b) a: Contains a but not $b \to$ rejected
(c) b: Contains b but not $a \to$ rejected
(d) ab: Contains both a and $b \to$ accepted
(e) ba: Contains both a and $b \to$ accepted
(f) $aabb$: Contains both a and $b \to$ accepted

8.2.3 Complement

$$L(M)^c = \Sigma^* \setminus L(M) = \{x \in \Sigma^* \mid x \notin L(M)\}$$

Given $M = (W, \Sigma, V, S, F)$, the complement is constructed as:

$$M^c = (W, \Sigma, V, S, F^c)$$

where

$$F^c = W \setminus F$$

The complement DFA has the same structure but with accepting and non-accepting states swapped.

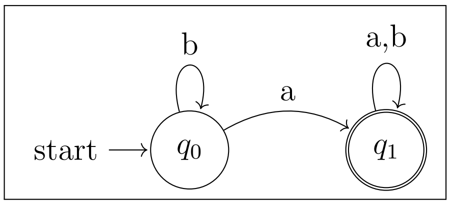
Example:

Let $L(M_1)$ be the set of strings over $\{a, b\}$ that contain at least one a. The complement language is:

$$L(M_1)^c = \{w \in \{a, b\}^* \mid w \text{ contains no } a\} = b^*.$$

Original DFA M_1: Accepts strings with at least one a.
$M_1 = (W, \Sigma, V, S, F)$ where

$$W = \{q_0, q_1\}, \quad \Sigma = \{a, b\}, \quad S = q_0, \quad F = \{q_1\}.$$

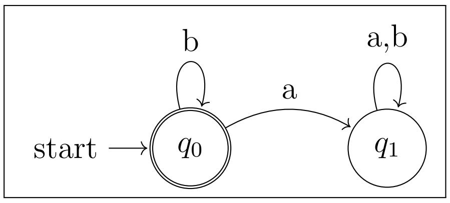


Complement Construction:

$$M_1^c = (W, \Sigma, V, S, F^c)$$

where

$$F^c = W \setminus F = \{q_0, q_1\} \setminus \{q_1\} = \{q_0\}.$$



The complement DFA accepts all strings that the original rejects, and vice versa. State q_0 (previously non-accepting) is now accepting, while state q_1 (previously accepting) is now non-accepting.

Thus,

$$L(M_1)^c = L(M_1^c) = b^* = \{\varepsilon, b, bb, bbb, \ldots\}.$$

Verification:

1. ε: No $a \to$ accepted
2. b: No $a \to$ accepted
3. a: Contains $a \to$ rejected
4. ab: Contains $a \to$ rejected
5. bbb: No $a \to$ accepted

8.2.4 Difference

$$L(M_1) - L(M_2) = \{x \in \Sigma^* \mid x \in L(M_1) \wedge x \notin L(M_2)\}$$

The difference is constructed using intersection with complement:

$$L(M_1) - L(M_2) = L(M_1) \cap L(M_2)^c$$

Given $M_1 = (W_1, \Sigma, V_1, S_1, F_1)$ and $M_2 = (W_2, \Sigma, V_2, S_2, F_2)$:

1. First, construct $M_2^c = (W_2, \Sigma, V_2, S_2, F_2^c)$ where $F_2^c = W_2 \setminus F_2$
2. Then, construct the product: $M = (W_1 \times W_2, \Sigma, V, (S_1, S_2), F)$ where

$$F = \{(w_1, w_2) \mid w_1 \in F_1 \wedge w_2 \in F_2^c\}$$

Example:

Let $L(M_1)$ be the set of strings over $\{a, b\}$ that contain at least one a, and let $L(M_2)$ be the set of strings that contain at least one b.

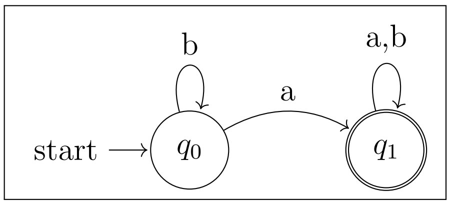
The difference language is:

$$L(M_1) - L(M_2) = \{w \in \{a, b\}^* \mid w \text{ contains at least one } a \text{ and contains no } b\} = a^+.$$

1. Component DFA M_1: Accepts strings with at least one a.

   $M_1 = (W_1, \Sigma, V_1, S_1, F_1)$ where

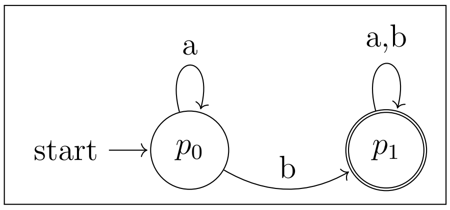
$$W_1 = \{q_0, q_1\}, \quad \Sigma = \{a, b\}, \quad S_1 = q_0, \quad F_1 = \{q_1\}.$$



2. Component DFA M_2: Accepts strings with at least one b.

   $M_2 = (W_2, \Sigma, V_2, S_2, F_2)$ where

$$W_2 = \{p_0, p_1\}, \quad \Sigma = \{a, b\}, \quad S_2 = p_0, \quad F_2 = \{p_1\}.$$



Difference Construction:

First, construct the complement of M_2:

$$M_2^c = (W_2, \Sigma, V_2, S_2, F_2^c) \quad \text{where} \quad F_2^c = W_2 \setminus F_2 = \{p_0, p_1\} \setminus \{p_1\} = \{p_0\}.$$

Then construct the product automaton:

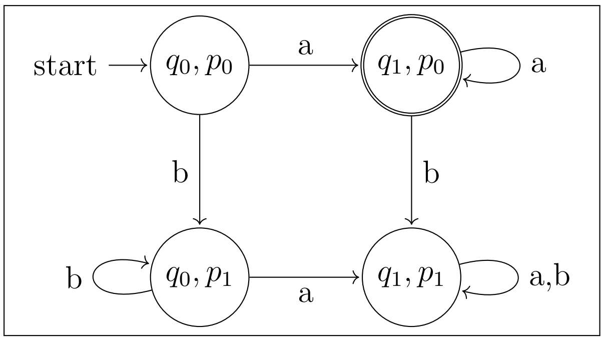
$$M = M_1 - M_2 = (W_1 \times W_2,\ \Sigma,\ V,\ (q_0, p_0),\ F)$$

where

$$W = \{(q_0, p_0), (q_0, p_1), (q_1, p_0), (q_1, p_1)\}$$

and

$$F = \{(x, y) \mid x \in F_1 \wedge y \in F_2^c\} = \{(q_1, p_0)\}.$$



The product state (q_i, p_j) represents "M_1 in state q_i and M_2^c in state p_j". Only state (q_1, p_0) is accepting because it represents strings where M_1 accepts (contains a) AND M_2^c accepts (contains no b).

Thus,

$$L(M_1) - L(M_2) = L(M) = a^+ = \{a, aa, aaa, \ldots\}.$$

Verification:

(a) ε: No $a \to$ rejected

(b) a: Contains a, no $b \to$ accepted

(c) b: Contains $b \to$ rejected

(d) ab: Contains $b \to$ rejected

(e) aaa: Contains a, no $b \to$ accepted

8.2.5 Concatenation

$$L(M_1) \cdot L(M_2) = \{xy \mid x \in L(M_1) \wedge y \in L(M_2)\}$$

Given DFAs $M_1 = (W_1, \Sigma, V_1, S_1, F_1)$ and $M_2 = (W_2, \Sigma, V_2, S_2, F_2)$, concatenation requires constructing an NFA (with ε-transitions):

$$M = M_1 \cdot M_2 = (W_1 \cup W_2,\ \Sigma,\ V,\ S_1,\ F_2)$$

where the transition function δ includes:

1. All transitions from M_1: $\delta(q, a) = V_1(q, a)$ for $q \in W_1$
2. All transitions from M_2: $\delta(q, a) = V_2(q, a)$ for $q \in W_2$
3. ε-transitions from every accepting state of M_1 to the start state of M_2:

$$V(f, \varepsilon) = S_2 \text{ for all } f \in F_1$$

Example:

Let $L(M_1)$ be the set of strings over $\{x, y\}$ that contain at least one x, and let $L(M_2)$ be the set of strings over $\{x, y\}$ that contain at least one y.

The concatenation language is:

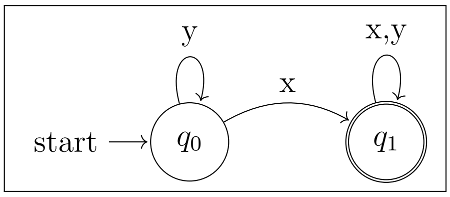
$$L(M_1) \cdot L(M_2) = \{uv \mid u \in L(M_1),\ v \in L(M_2)\}$$

This represents all strings that can be split into two parts: the first part contains at least one x, and the second part contains at least one y.

1. Component DFA M_1: Accepts strings with at least one x.

   $M_1 = (W_1, \Sigma, V_1, S_1, F_1)$ where

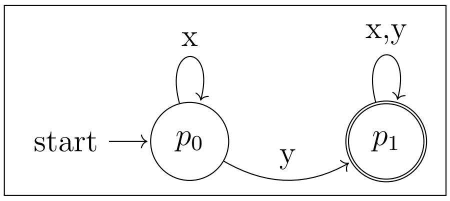
$$W_1 = \{q_0, q_1\}, \quad \Sigma = \{x, y\}, \quad S_1 = q_0, \quad F_1 = \{q_1\}.$$



2. Component DFA M_2: Accepts strings with at least one y.

   $M_2 = (W_2, \Sigma, V_2, S_2, F_2)$ where

$$W_2 = \{p_0, p_1\}, \quad \Sigma = \{x, y\}, \quad S_2 = p_0, \quad F_2 = \{p_1\}.$$

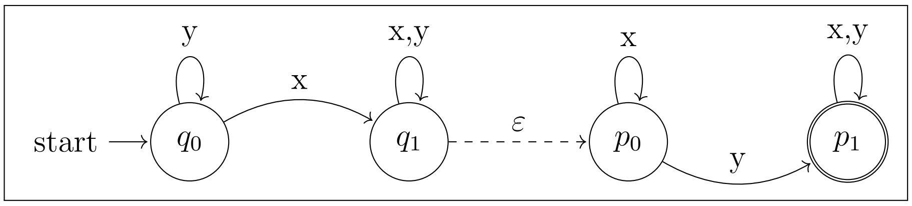


Concatenation Construction (NFA):

$$M = M_1 \cdot M_2 = (W_1 \cup W_2,\ \Sigma,\ V,\ q_0,\ F_2)$$

where $W = \{q_0, q_1, p_0, p_1\}$, start state is q_0, accepting states are $F = \{p_1\}$, and the ε-transition is:

$$q_1 \xrightarrow{\varepsilon} p_0$$



The ε-transition allows the NFA to non-deterministically "jump" from the accepting state of M_1 to the start state of M_2, effectively concatenating the two languages.

Thus,

$$L(M_1 \cdot M_2) = L(M_1)L(M_2).$$

Verification:

(a) xy: First part $x \in L(M_1)$, second part $y \in L(M_2) \to$ accepted

(b) xxy: First part $xx \in L(M_1)$, second part $y \in L(M_2) \to$ accepted

(c) xyy: First part $x \in L(M_1)$, second part $yy \in L(M_2) \to$ accepted

(d) $xxyy$: Multiple valid splits $\to$ accepted

(e) x: No part with $y \to$ rejected

(f) y: No part with $x \to$ rejected

Unlike the previous closure properties (union, intersection, complement, difference), concatenation requires an NFA with ε-transitions. DFAs alone cannot directly construct concatenation, though the resulting NFA can be converted to a DFA using standard NFA-to-DFA conversion.

8.2.6 Kleene Star

Example:

Let $L(M_1)$ be the set of strings over $\{p, q\}$ that contain at least one p.

The Kleene star language is:

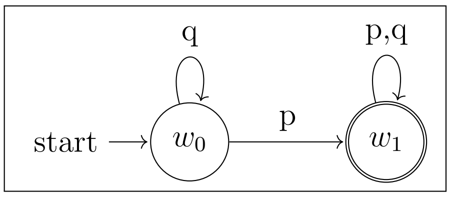
$$L(M_1)^* = \{w_1 w_2 \cdots w_k \mid k \geq 0,\ \forall i,\ w_i \in L(M_1)\}.$$

Since each w_i must contain at least one p, any string in $L(M_1)^*$ is a concatenation of blocks each containing a p.

$$L(M_1)^* = \{p, q\}^* \setminus \{q^n \mid n \geq 1\}.$$

1. $M_1 = (W_1, \Sigma, V_1, S_1, F_1)$

$$W_1 = \{w_0, w_1\}, \quad \Sigma = \{p, q\}, \quad S_1 = w_0, \quad F_1 = \{w_1\}.$$

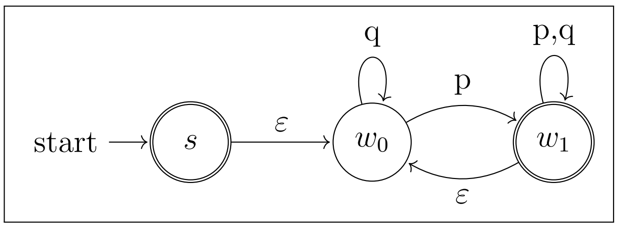


To build M_1^*, we create a new start/accepting state s and add:

$$s \xrightarrow{\varepsilon} S_1, \qquad F_1 \xrightarrow{\varepsilon} S_1.$$

So the NFA is:

$$M_1^* = (W_1 \cup \{s\},\ \Sigma,\ V,\ s,\ F_1 \cup \{s\}).$$



Thus,

$$L(M_1)^* = L(M_1^*).$$

8.2.7 Reversal

Example:

Let $L(M_1)$ be the set of strings over $\{s, a\}$ that end with symbol a.
The reversal language is:

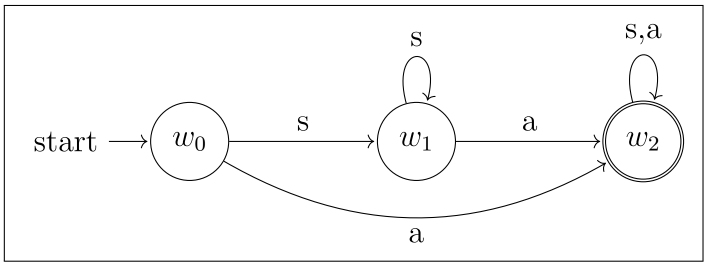
$$L(M_1)^R = \{w^R \mid w \in L(M_1)\}.$$

Since w ends with a, the reversed string w^R must begin with a.
Thus,

$$L(M_1)^R = \{a\{s, a\}^*\}.$$

1. $M_1 = (W_1, \Sigma, V_1, S_1, F_1)$

$$W_1 = \{w_0, w_1, w_2\}, \quad \Sigma = \{s, a\}, \quad S_1 = w_0, \quad F_1 = \{w_2\}.$$



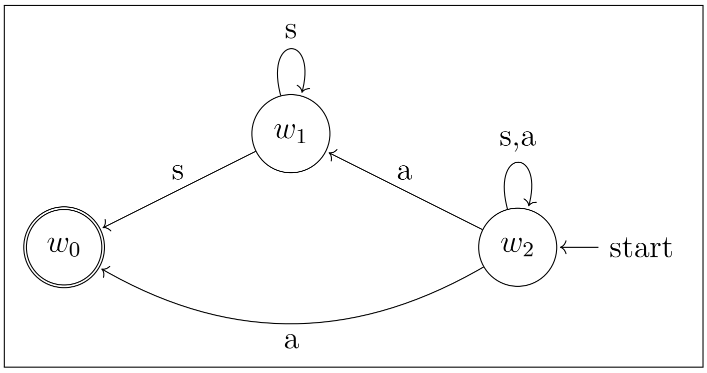
To construct M_1^R:

(a) Reverse all transitions.
(b) Make original final states into the new start states.
(c) Make the original start state the only accepting state.
(d) Because multiple ε-entries may arise, the construction yields an NFA.

Hence,

$$M_1^R = (W_1,\ \Sigma,\ V^R,\ F_1,\ \{S_1\}).$$

Where V^R is the reverse of V_1.



Therefore,

$$L(M_1)^R = L(M_1^R).$$

8.2.8 Homomorphism

Example:

Let $L(M_1)$ be the set of strings over $\{0,1\}$ that contain at least one 1:

$$L(M_1) = \{w \in \{0,1\}^* \mid w \text{ contains at least one } 1\}.$$

Define a homomorphism h:

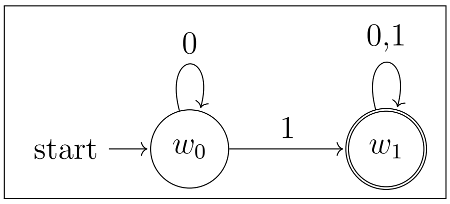
$$h(0) = 01, \quad h(1) = 1.$$

Then the homomorphic image of $L(M_1)$ is:

$$h(L(M_1)) = \{h(w) \mid w \in L(M_1)\}.$$

Let
$M_1 = (W_1, \Sigma, V_1, S_1, F_1)$

$$W_1 = \{w_0, w_1\}, \quad \Sigma = \{0,1\}, \quad S_1 = w_0, \quad F_1 = \{w_1\}.$$



The homomorphism h transforms each 0 into 01, and each 1 into 1. For example:

$$w = 0101 \in L(M_1) \to h(w) = 01 \cdot 01 \cdot 1 \cdot 1 = 010111 \in h(L(M_1)).$$

Hence,

$$h(L(M_1)) \text{ is regular.}$$

8.2.9 Inverse Homomorphism

Example:

Let $L(M_2)$ be the set of strings over $\{0,1\}$ that start with 1:

$$L(M_2) = \{w \in \{0,1\}^* \mid w \text{ starts with } 1\}.$$

Define a homomorphism h:

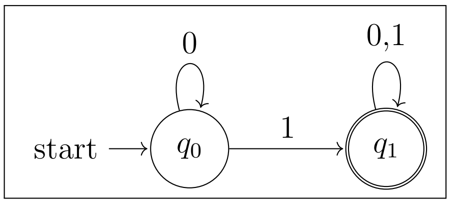
$$h(0) = 01, \quad h(1) = 1.$$

The inverse homomorphic image of $L(M_2)$ is:

$$h^{-1}(L(M_2)) = \{w \in \{0,1\}^* \mid h(w) \in L(M_2)\}.$$

Let
$M_2 = (W_2, \Sigma, V_2, S_2, F_2)$

$$W_2 = \{q_0, q_1\}, \quad \Sigma = \{0,1\}, \quad S_2 = q_0, \quad F_2 = \{q_1\}.$$



The inverse homomorphism h^{-1} gives the set of strings w such that $h(w)$ starts with 1. For instance:

$$w = 0 \to h(0) = 01 \notin L(M_2) \to 0 \notin h^{-1}(L(M_2)),$$
$$w = 1 \to h(1) = 1 \in L(M_2) \to 1 \in h^{-1}(L(M_2)).$$

Thus,

$$h^{-1}(L(M_2)) \text{ is regular.}$$

8.2.10 Substitution

Example:

Let $L(M_3)$ be the set of strings over $\{0,1\}$ that contain exactly one 1:

$$L(M_3) = \{w \in \{0,1\}^* \mid w \text{ contains exactly one } 1\}.$$

Define a substitution ϕ:

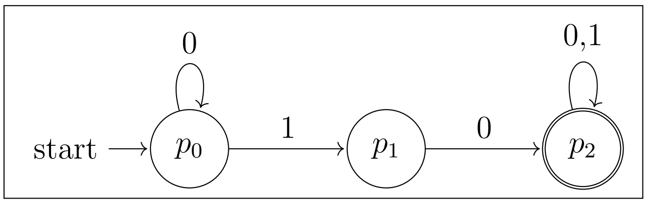
$$\phi(0) = \{0, 00\}, \quad \phi(1) = \{1, 11\}.$$

The substitution of $L(M_3)$ is:

$$\phi(L(M_3)) = \bigcup_{w \in L(M_3)} \phi(w),$$

where $\phi(w)$ replaces each symbol x in w by a string from $\phi(x)$.
Let
$M_3 = (W_3, \Sigma, V_3, S_3, F_3)$

$$W_3 = \{p_0, p_1, p_2\}, \quad \Sigma = \{0,1\}, \quad S_3 = p_0, \quad F_3 = \{p_2\}.$$



The substitution allows each 0 to be replaced by 0 or 00, and each 1 by 1 or 11, producing a regular language:

$$\phi(L(M_3)) \text{ is regular.}$$

9 Pumping Lemma

For any regular language L, there exists a constant $p \geq 1$ (called the pumping length) such that for any string $w \in L$ with $|w| \geq p$, there exists a decomposition $w = xyz$ satisfying:

$$|y| > 0,\ |xy| \leq p,\ \forall i \geq 0,\ xy^iz \in L$$

Example:
Let

$$A = \{a^nb^n \mid n \geq 0\} \text{ is not regular}$$

Proof
Assume A is regular. Let p be the pumping length. Consider $w = a^pb^p \in A$ with $|w| = 2p \geq p$. By the Pumping Lemma, $w = xyz$ where $|y| > 0$, $|xy| \leq p$, and $\forall i \geq 0, xy^iz \in A$. Since $|xy| \leq p$, both x and y consist only of a's. Let $y = a^k$ where $k > 0$. Pump with $i = 2$: $xy^2z = a^{p+k}b^p$. Since $k > 0$, we have $p + k > p$, so $xy^2z \notin A$. Contradiction. Therefore, A is not regular. □

Intuitively:
Suppose $p = 7$. Consider $w = a^7b^7$ where $|xy| \leq 7$. There are three cases for where y can be located:

1. y is in the a part.
   Let $w = \underbrace{aa}_{x}\underbrace{aaaa}_{y}\underbrace{ab^7}_{z}$.
   Pumping with $i = 2$: $xy^2z = a^{11}b^7 \notin A$ since $11 \neq 7$.
2. y is in the b part.
   Let $w = \underbrace{a^7}_{x}\underbrace{bb}_{y}\underbrace{b^5}_{z}$.
   Pumping with $i = 2$: $xy^2z = a^7b^9 \notin A$ since $7 \neq 9$.
3. y spans both a and b parts.
   Let $w = \underbrace{aaaa}_{x}\underbrace{aaab}_{y}\underbrace{b^6}_{z}$.
   Pumping with $i = 2$: $xy^2z = aaaaaaabaabb^6 \notin A$ (wrong pattern for a^nb^n).

Since $|xy| \leq p = 7$, all cases lead to contradiction. Therefore, A is not regular.

10 Regular Grammar

Formally

$$G = (V, T, S, P)$$

where:

1. V = a finite set of variables (nonterminal symbols),

2. T = a finite set of terminal symbols,
3. S = the start symbol ($S \in V$),
4. P = the set of production rules for terminals and nonterminals.

In general:
Let

$$G = (\{f, g, h\},\ \{0, 1\},\ f,\ \{f \to gh,\ g \to 0,\ h \to 1\})$$

where

$$\begin{aligned} V &= \{f, g, h\} & T &= \{0, 1\}, \\ S &= f & P &= \{\, f \to gh,\ g \to 0,\ h \to 1 \,\}. \end{aligned}$$

From this we can derive

$$\begin{aligned} f &\to gh \\ &\to 0h \\ &\to 01 \end{aligned}$$

Hence, we can coclude,

$$\boxed{\text{L(G)} = \{10\}}$$

11 Context-Free Grammar and Language

Formally, a context-free grammar is $G = (V, \Sigma, R, S)$ where

1. V is the set of nonterminals.
2. Σ is the set of terminals.
3. $S \in V$ is the start symbol.
4. $R \subseteq V \times (V \cup \Sigma)^*$ is the set of productions, each of the form $A \to \alpha$ with $A \in V$ and $\alpha \in (V \cup \Sigma)^*$.

To produce strings, a CFG uses the operations of concatenation and union.

1. **Concatenation:** If $A \to \alpha$ and $B \to \beta$, then writing $\alpha\beta$ denotes placing the two strings next to each other.
2. **Union:** A production such as $A \to xA \mid x$ means A may produce either xA or x.

Example:
Consider the grammar

$$G = (\{f, g, h\}, \{x, y\}, f,\ \{f \to gh,\ g \to xg \mid x,\ h \to yh \mid y\}).$$

This grammar generates strings consisting of one or more x's followed by one or more y's.

$$\begin{aligned} f &\to \underline{g}h && [f \to gh] \\ &\to xh && [g \to x] \\ &\to xy && [h \to y] \end{aligned}$$

$$\begin{aligned} f &\to \underline{g}h && [f \to gh] \\ &\to x\underline{g}h && [g \to xg] \\ &\to xxh && [g \to x] \\ &\to xxy && [h \to y] \end{aligned}$$

$$\begin{aligned} f &\to g\underline{h} && [f \to gh] \\ &\to gy\underline{h} && [h \to yh] \\ &\to gyy && [h \to y] \\ &\to xyy && [g \to x] \end{aligned}$$

$$\begin{aligned} f &\to \underline{g}h && [f \to gh] \\ &\to x\underline{g}h && [g \to xg] \\ &\to xx\underline{h} && [g \to x] \\ &\to xxy\underline{h} && [h \to yh] \\ &\to xxyy && [h \to y] \end{aligned}$$

Therefore,

$$\boxed{L(G) = \{xy, x^2y, xy^2, x^2y^2, \dots\}}$$

More compactly,

$$\boxed{L(G) = \{x^m y^n \mid m \geq 1,\ n \geq 1\}}.$$

11.1 Recursive

Formally, a nonterminal $A \in V$ is called *recursive* if there exists a derivation

$$A \to^+ \alpha A \beta$$

for some $\alpha, \beta \in (V \cup \Sigma)^*$.

1. **Example 1:**

   Let the grammar

   $$G = (\{f\},\ \{0,1\},\ f,\ R)$$

   with productions

   $$R = \{\, f \to 01, \quad f \to 0f1 \,\}.$$

   We can derive:

   $$\begin{aligned} f &\to 0\underline{f}1 && [f \to 0f1] \\ &\to 0011 && [f \to 01] \end{aligned}$$

   $$\begin{aligned} f &\to 0\underline{f}1 && [f \to 0f1] \\ &\to 00\underline{f}11 && [f \to 0f1] \\ &\to 000111 && [f \to 01] \end{aligned}$$

   Thus the language is

   $$\boxed{L(G) = \{\, 0^n 1^n \mid n \geq 1 \,\}}$$

2. **Example 2:**

   Let
$$G = (\{f, g\},\ \{0, 1\},\ f,\ R)$$
   with productions
$$R = \{\, f \to 0g1, \quad g \to 0g1, \quad g \to \varepsilon \,\}.$$
   Then we can derive:

$$\begin{aligned} f &\to 0\underline{g}1 && [\text{by } f \to 0g1] \\ &\to 0\underline{0g1}1 && [\text{by } g \to 0g1] \\ &\to 00\underline{0g1}11 && [\text{by } g \to 0g1] \\ &\to 000\underline{g}111 && [\text{by } g \to \varepsilon] \\ &= 000111 \end{aligned}$$

$$\begin{aligned} f &\to 0\underline{g}1 && [\text{by } f \to 0g1] \\ &\to 01 && [\text{by } g \to \varepsilon] \\ &= 01 \end{aligned}$$

$$\begin{aligned} f &\to 0\underline{g}1 && [\text{by } f \to 0g1] \\ &\to 0\underline{0g1}1 && [\text{by } g \to 0g1] \\ &\to 00\underline{g}11 && [\text{by } g \to \varepsilon] \\ &= 0011 \end{aligned}$$

$$\begin{aligned} f &\to 0\underline{g}1 && [\text{by } f \to 0g1] \\ &\to 0\underline{0g1}1 && [\text{by } g \to 0g1] \\ &\to 00\underline{0g1}11 && [\text{by } g \to 0g1] \\ &\to 0000\underline{g}1111 && [\text{by } g \to \varepsilon] \\ &= 00001111 \end{aligned}$$

   This grammar generates
$$\boxed{L(G) = \{0^n 1^n \mid n \geq 1\}}.$$
   Note that the grammar defines how to produce strings; the language is the complete set of strings producible from that grammar.

3. **Example 3:**

   Let
$$G = (\{f\},\ \{(,)\},\ f,\ R)$$
   with productions
$$R = \{\, f \to (f), \quad f \to ff, \quad f \to \varepsilon \,\}.$$
   A sample derivation of ():

$$\begin{aligned} f &\to (\underline{f}) && [f \to (f)] \\ &\to ((\underline{f})) && [f \to (f)] \\ &\to (((\underline{f}))) && [f \to (f)] \\ &\to ((())) && [f \to \varepsilon] \end{aligned}$$

   A sample derivation of ()():

$$
\begin{aligned}
f &\to \underline{f}f && [f \to ff] \\
&\to (\underline{f})f && [f \to (f)] \\
&\to ()f && [f \to \varepsilon] \\
&\to ()(\underline{f}) && [f \to (f)] \\
&\to ()() && [f \to \varepsilon]
\end{aligned}
$$

A sample derivation of $((()()))$:

$$
\begin{aligned}
f &\to (\underline{f}) && [f \to (f)] \\
&\to ((\underline{f})) && [f \to (f)] \\
&\to ((\underline{f}f)) && [f \to ff] \\
&\to ((()\underline{f})) && [f \to (f)] \\
&\to ((()())) && [f \to (f),\ \text{then } f \to \varepsilon] \\
&\to ((()())) && [f \to \varepsilon]
\end{aligned}
$$

Therefore,

$$\boxed{L(G) = \text{all balanced parentheses strings}}.$$

4. **Example 4:**

   Let

   $$G = (\{g\},\ \{+, *, (,), a\},\ g,\ R)$$

   with productions

   $$R = \{\, g \to g + g, \quad g \to g * g, \quad g \to (g), \quad g \to a \,\}.$$

   A sample derivation of $(a + a) * a$:

   $$
   \begin{aligned}
   g &\to \underline{g} * g && [g \to g * g] \\
   &\to (\underline{g}) * g && [g \to (g)] \\
   &\to (\underline{g} + g) * g && [g \to g + g] \\
   &\to (a + \underline{g}) * g && [g \to a] \\
   &\to (a + a) * \underline{g} && [g \to a] \\
   &\to (a + a) * a && [g \to a]
   \end{aligned}
   $$

   Hence,

   $$\boxed{L(G) = \text{the set of all well-formed arithmetic expressions over } a}$$

   (with operations $+, *, ()$).

11.2 Right and Left Linear Grammar

1. Right Linear (nonterminal on the right):

$$g \to 1h \quad \text{(produces strings like } 111\ldots \text{ from left to right)}$$

   Example:
   Let
   $$G_R = (\{f, g\},\ \{1, 0\},\ f,\ R_R)$$
   with productions
   $$R_R = \{\, f \to 1g,\ g \to 1g,\ g \to 0 \,\}.$$
   This generates strings such as
   $$f \to 1g \to 11g \to 111g \to 1110.$$

2. Left Linear (nonterminal on the left):

$$g \to h1 \quad \text{(produces strings like } \ldots 111 \text{ from right to left)}$$

Example:
Let
$$G_L = (\{f, g\},\ \{1, 0\},\ f,\ R_L)$$
with productions
$$R_L = \{\, f \to g1,\ g \to g1,\ g \to 0 \,\}.$$
This generates the same language, but the steps look reversed:
$$f \to g1 \to g11 \to g111 \to 0111.$$

11.3 Pumping Lemma for Contex-Free Languages

For any context-free language L, there exists a constant $p \geq 1$ such that for any string $w \in L$ with $|w| \geq p$, there exists a decomposition $w = uvxyz$ satisfying:

$$|vy| > 0,\ |vxy| \leq p,\ \forall i \geq 0,\ uv^i xy^i z \in L$$

Example: Let

$$A = \{f^n g^n h^n \mid n \geq 0\} \text{ is not context-free}$$

Proof
Assume A is context-free. Let p be the pumping length. Consider $w = f^p g^p h^p \in A$ with $|w| = 3p \geq p$. By the Pumping Lemma, $w = uvxyz$ where $|vy| > 0$, $|vxy| \leq p$, and $\forall i \geq 0,\ uv^i xy^i z \in A$. Since $|vxy| \leq p$ and w contains p copies each of f, g, and h, the substring vxy can contain at most two distinct symbols. Therefore, v and y together can only pump at most two of the three symbols f, g, h. Pump with $i = 2$: uv^2xy^2z will have unequal counts of f's, g's, and h's. Thus $uv^2xy^2z \notin A$. Contradiction. Therefore, A is not context-free. □

Intuitively:

Suppose $p = 4$. Consider $w = f^4g^4h^4$ where $|vxy| \leq 4$. There are several cases for where vxy can be located:

1. vxy is in the f part only.

   Let $w = \underbrace{}_{u}\underbrace{f}_{v}\underbrace{ff}_{x}\underbrace{f}_{y}\underbrace{g^4h^4}_{z}$.

   Pumping with $i = 2$: $uv^2xy^2z = f^6g^4h^4 \notin A$ since $6 \neq 4 \neq 4$.

2. vxy spans the f and g parts.

   Let $w = \underbrace{ff}_{u}\underbrace{f}_{v}\underbrace{fg}_{x}\underbrace{g}_{y}\underbrace{g^2h^4}_{z}$.

   Pumping with $i = 2$: $uv^2xy^2z = f^5g^5h^4 \notin A$ since $5 \neq 4$.

3. vxy is in the g part only.

   Let $w = \underbrace{f^4}_{u}\underbrace{g}_{v}\underbrace{gg}_{x}\underbrace{g}_{y}\underbrace{h^4}_{z}$.

   Pumping with $i = 2$: $uv^2xy^2z = f^4g^6h^4 \notin A$ since $4 \neq 6 \neq 4$.

4. vxy spans the g and h parts.

   Let $w = \underbrace{f^4g^2}_{u}\underbrace{g}_{v}\underbrace{gh}_{x}\underbrace{h}_{y}\underbrace{h^2}_{z}$.

   Pumping with $i = 2$: $uv^2xy^2z = f^4g^5h^5 \notin A$ since $4 \neq 5$.

5. vxy is in the h part only.

   Let $w = \underbrace{f^4g^4}_{u}\underbrace{h}_{v}\underbrace{hh}_{x}\underbrace{h}_{y}\underbrace{}_{z}$.

   Pumping with $i = 2$: $uv^2xy^2z = f^4g^4h^6 \notin A$ since $4 \neq 6$.

Since $|vxy| \leq p = 4$, the substring vxy can span at most two of the three symbol types. In all cases, pumping causes unequal counts. Therefore, A is not context-free.

12 Derivation Tree

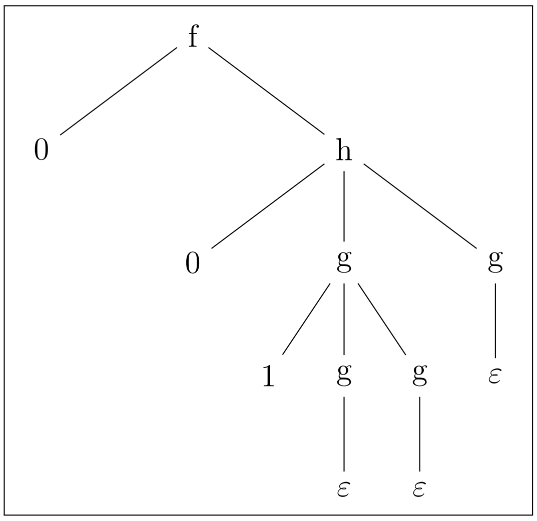
A Derivation Tree or Parse Tree is an ordered rooted tree that graphically represents the semantic information of strings derived from a Context-free Grammar.

Example:

Given the grammar:

$$G_T = \{f \to 0h, g \to 1gg \mid \varepsilon, h \to 0gg\}$$

Derivation tree for the string "010":



12.1 Right and Left Derivation

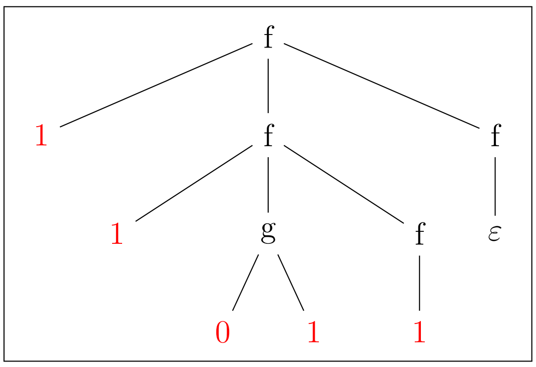
Example:

Given the grammar:

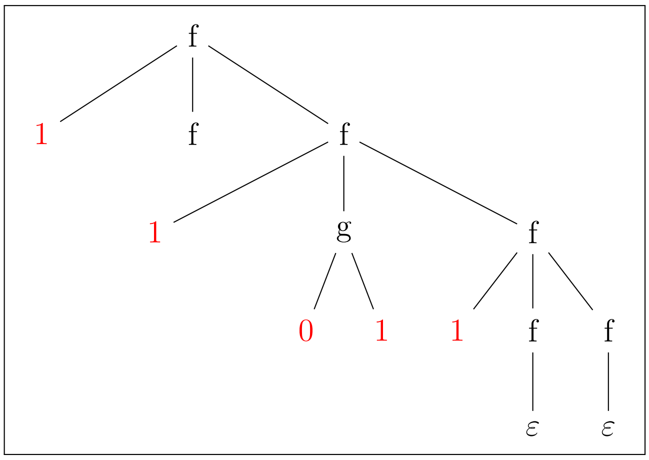
$$G_T = \{f \to 1gf \mid 1ff \mid \varepsilon, g \to f0g \mid 01\}$$

Generate the string "11011":

1. Left derivation



2. Right derivation:



12.2 Ambiguous Grammar

A grammar is said to be ambiguous if there exist two or more derivation tree for a string

Example:

Let

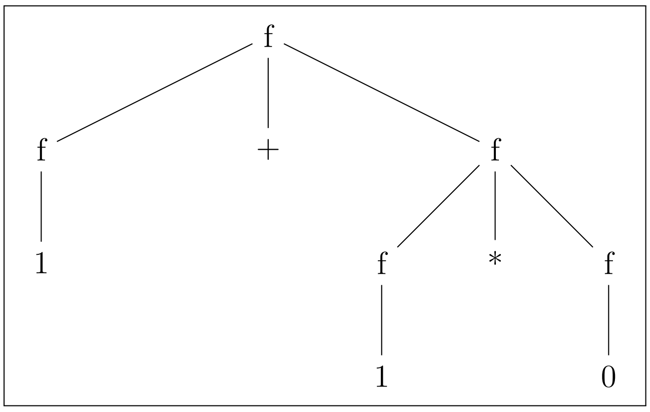
$$G = (\{f\}, \{1, 0, +, *\}, f, \{f \to f + f \mid f * f \mid 1 \mid 0\})$$

This grammar is ambiguous because we can derive the string $1+(1*0)$ in two different ways:

1. Derivation 1 (Addition at root)

$$\begin{aligned} f &\to f + f \\ &\to 1 + f \\ &\to 1 + f * f \\ &\to 1 + 1 * f \\ &\to 1 + 1 * 0 \end{aligned}$$

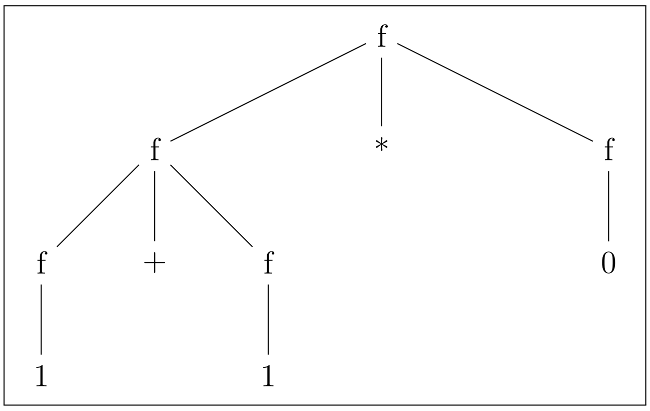
This corresponds to the parse tree: $1 + (1 * 0)$ where multiplication has higher precedence.



2. Derivation 2 (Multiplication at root)

$$\begin{aligned} f &\to f * f \\ &\to f + f * f \\ &\to 1 + f * f \\ &\to 1 + 1 * f \\ &\to 1 + 1 * 0 \end{aligned}$$

This corresponds to the parse tree: $(1+1)*0$ where addition has higher precedence.



Since the same string $1+(1*0)$ can be derived with two different parse trees (different structural interpretations), the grammar is ambiguous. This ambiguity arises because the grammar doesn't specify operator precedence or associativity rules.

13 Simplification of CFG

13.1 Reduction of CFG

A symbol $X \in (V \cup T)$ is useful if:

1. X is generating: $X \to^* w$ where $w \in T^*$
2. X is reachable: $S \to^* \alpha X \beta$ where $\alpha, \beta \in (V \cup T)^*$

Algorithm:

1. Eliminate non-generating symbols
2. Eliminate unreachable symbols

Example:
Given

$$P = \{S \to AC \mid B, A \to a, C \to c \mid BC, E \to aA \mid e\}$$

1. Step 1: Eliminate non-generating symbols

   (a) Generating symbols:

   $$S \to AC, A \to a, C \to c, E \to e, E \to aA$$

   (b) B is non-generating (no production for B)

   (c) Remove productions with

   $$B : P_1 = \{S \to AC, A \to a, C \to c, E \to aA \mid e\}$$

2. Step 2: Eliminate unreachable symbols

   (a) From $S \to AC$: reachable symbols are S, A, C

   (b) E is unreachable from S

   (c) Remove unreachable symbols:

   $$P_2 = \{S \to AC, A \to a, C \to c\}$$

Hence, the reduced grammar is:

$$G' = (\{S, A, C\}, \{a, c\}, S, \{S \to AC, A \to a, C \to c\})$$

13.2 Removal Unit of Production

A production $A \to B$ is a unit production if $A, B \in V$ (both are variables).
Algorithm:

1. Find all unit productions
2. For each unit production $A \to B$, replace it with $A \to \alpha$ for all non-unit productions $B \to \alpha$
3. Remove all unit productions

Example:
Given

$$P = \{S \to XY, X \to a, Y \to Z \mid b, Z \to M, M \to N, N \to a\}$$

1. Step 1: Identify unit productions

   $$Y \to Z, \quad Z \to M, \quad M \to N$$

2. Step 2: Find unit closure for each variable

   (a) Since $N \to a$, we add $M \to a$
   (b) Since $M \to a$, we add $Z \to a$
   (c) Since $Z \to a$, we add $Y \to a$

   Then

$$P = \{S \to XY, X \to a, Y \to a \mid b, Z \to a, M \to a, N \to a\}$$

   Furthermore, we remove the unreachable symbols, that is M, N, and Z.

Therefore, the reduced grammar is:

$$G' = (\{S, X, Y\}, \{a, b\}, \{S \to XY, X \to a, Y \to a \mid b\}, S)$$

13.3 Removal of Null Productions

A production $A \to \varepsilon$ is a null production (or ε-production).
Algorithm:

1. Find all nullable variables (variables that can derive ε)
2. For each production, generate new productions by removing nullable variables in all possible combinations
3. Remove all null productions

Example:

$$P = \{S \to ABAC, A \to aA \mid \varepsilon, B \to bB \mid \varepsilon, C \to c\}$$

1. Step 1: Identify nullable variables
   (a) $A \to \varepsilon$, so A is nullable
   (b) $B \to \varepsilon$, so B is nullable

   Thus, nullable variables: $\{A, B\}$

2. Step 2: Generate new productions by removing nullable variables
   (a) To Eliminate $A \to \varepsilon$

$$\begin{aligned} S &\to ABAC \\ &\to ABC \\ &\to BAC \\ &\to BC \\ &= S \to ABAC \mid ABC \mid BAC \mid BC \end{aligned} \qquad \begin{aligned} &A \to aA \\ &A \to a \\ &\quad = A \to aA \mid a \end{aligned}$$

   Then, the new production:

$$\begin{aligned} P_1 = \{&S \to ABAC \mid ABC \mid BAC \mid BC, \\ &A \to aA \mid a, B \to bB \mid \varepsilon, C \to c\} \end{aligned}$$

(b) To Eliminate $B \to \varepsilon$

$$\begin{aligned} &S \to AAC \mid AC \mid C \\ &B \to b \end{aligned}$$

Hence, the new production:

$$\begin{aligned} P_2 = \{&S \to ABAC \mid ABC \mid BAC \mid BC \mid AAC \mid AC \mid C, \\ &A \to aA \mid a, B \to bB \mid b, C \to c\} \end{aligned}$$

14 Chomsky Normal Form (CNF)

A CFG is in CNF if every production rule has one of these forms:

1. $A \to BC$, where $A, B, C \in V$ (non-terminals)
2. $A \to a$, where $A \in V$ and $a \in \Sigma$ (terminal)
3. $S \to \varepsilon$, where S is the start symbol and S does not appear on the right-hand side of any production

Example:
Consider the following grammar in CNF:

$$G = (V, \Sigma, R, S)$$

where:

$$V = \{S, A, B, C\}, \quad \Sigma = \{a, b\}$$

Production Rules:

$$\begin{aligned} &S \to AB, && \text{(Rule 1: two non-terminals)} \\ &S \to BC, && \text{(Rule 1: two non-terminals)} \\ &A \to a, && \text{(Rule 2: single terminal)} \\ &B \to b, && \text{(Rule 2: single terminal)} \\ &C \to SA, && \text{(Rule 1: two non-terminals)} \\ &C \to a, && \text{(Rule 2: single terminal)} \end{aligned}$$

14.1 CFG to CNF

There are 5 steps involved in the conversion of CFG to Chomsky Normal Form:

1. If the start symbol ϕ occurs on the right-hand side of any production rule, create a new start symbol ϕ' and add a new production rule $\phi' \to \phi$.
2. Remove null productions (productions of the form $\chi \to \varepsilon$).
3. Remove unit productions (productions of the form $\chi \to \eta$ where $\chi, \eta \in V$).

4. Replace each production $\chi \to \eta_1\eta_2\ldots\eta_n$ where $n > 2$ with:

$$\begin{aligned} \chi &\to \eta_1 L_1 \\ L_1 &\to \eta_2 L_2 \\ L_2 &\to \eta_3 L_3 \\ &\vdots \\ L_{n-2} &\to \eta_{n-1}\eta_n \end{aligned}$$

   where $L_1, L_2, \ldots, L_{n-2}$ are new non-terminals.

5. Replace each production $\chi \to \alpha$ where α contains terminals mixed with non-terminals with:
   (a) For each terminal a in α, create a new non-terminal T_a and production $T_a \to a$
   (b) Replace a with T_a in the original production

Example:
Given

$$P = \{f \to \chi f \chi \mid 1\eta,\ \chi \to \eta \mid f,\ \eta \to 0 \mid \varepsilon\}$$

1. Step 1: Since f appears in RHS, we add a new start symbol f', and add $f' \to f$
   Hence,
   $$P = \{f' \to f,\ f \to \chi f \chi \mid 1\eta,\ \chi \to \eta \mid f,\ \eta \to 0 \mid \varepsilon\}$$
2. Step 2: Remove the null productions, that is $\eta \to \varepsilon$
   First, identify all nullable symbols:
   (a) η is nullable (has production $\eta \to \varepsilon$)
   (b) χ is nullable (has production $\chi \to \eta$, and η is nullable)

   (a) After removing $\eta \to \varepsilon$
   $$f' \to f,\ f \to \chi f \chi \mid 1\eta \mid 1 \mid \chi f \mid f\chi \mid f,\ \chi \to \eta \mid f \mid \varepsilon,\ \eta \to 0$$
   (b) After removing $\chi \to \varepsilon$
   $$f' \to f,\ f \to \chi f \chi \mid f\chi \mid \chi f \mid f \mid 1\eta \mid 1,\ \chi \to \eta \mid f,\ \eta \to 0$$
3. Step 3: Remove unit productions: $f' \to f$, $f \to f$, $\chi \to \eta$, and $\chi \to f$
   (a) For $f' \to f$: Replace with what f produces:
   $$f' \to \chi f \chi \mid f\chi \mid \chi f \mid 1\eta \mid 1$$
   (b) For $f \to f$: Remove it
   $$f \to \chi f \chi \mid f\chi \mid \chi f \mid 1\eta \mid 1$$

(c) For $\chi \to \eta$: Replace with what η produces:

$$\chi \to 0 \mid f$$

(d) For $\chi \to f$: Replace with what f produces:

$$\chi \to \chi f \chi \mid f\chi \mid \chi f \mid 1\eta \mid 1$$

After removing all unit productions:

$$\begin{aligned} f' &\to \chi f \chi \mid f\chi \mid \chi f \mid 1\eta \mid 1 \\ f &\to \chi f \chi \mid f\chi \mid \chi f \mid 1\eta \mid 1 \\ \chi &\to 0 \mid \chi f \chi \mid f\chi \mid \chi f \mid 1\eta \mid 1 \\ \eta &\to 0 \end{aligned}$$

4. Step 4: Replace productions with more than 2 non-terminals

   For $f' \to \chi f \chi$, $f \to \chi f \chi$, and $\chi \to \chi f \chi$: introduce L_1, so:

$$\begin{aligned} f' &\to \chi L_1 \\ f &\to \chi L_1 \\ \chi &\to \chi L_1 \\ L_1 &\to f\chi \end{aligned}$$

   After replacing all such productions:

$$\begin{aligned} f' &\to \chi L_1 \mid f\chi \mid \chi f \mid 1\eta \mid 1 \\ f &\to \chi L_1 \mid f\chi \mid \chi f \mid 1\eta \mid 1 \\ \chi &\to 0 \mid \chi L_1 \mid f\chi \mid \chi f \mid 1\eta \mid 1 \\ \eta &\to 0 \\ L_1 &\to f\chi \end{aligned}$$

5. Step 5: Replace terminals in productions with non-terminals

   For productions like $f \to 1\eta$ and $f \to 1$, introduce $T_1 \to 1$:

   Final CNF:

$$\begin{aligned} f' &\to \chi L_1 \mid f\chi \mid \chi f \mid T_1\eta \mid 1 \\ f &\to \chi L_1 \mid f\chi \mid \chi f \mid T_1\eta \mid 1 \\ \chi &\to 0 \mid \chi L_1 \mid f\chi \mid \chi f \mid T_1\eta \mid 1 \\ \eta &\to 0 \\ L_1 &\to f\chi \\ T_1 &\to 1 \end{aligned}$$

15 Greibach Normal Form (GNF)

A CFG is in GNF if the productions are in the following forms:

$$\begin{aligned}&A \to b\\&A \to b f_1 f_2 \dots f_n\end{aligned}$$

where:

1. $A, f_1, f_2, \dots, f_n$ are non-terminal symbols
2. b is a terminal symbol
3. $n \geq 1$ (one or more non-terminals may follow the terminal)

Additionally, the start symbol S may have the production $S \to \varepsilon$ if ε is in the language.
In General:
Consider the following CFG:

$$\begin{aligned}&S \to AB \mid a\\&A \to aA \mid b\\&B \to b\end{aligned}$$

This grammar is not in GNF because:

1. $S \to AB$ starts with a non-terminal (not a terminal)
2. $B \to b$ is acceptable (starts with terminal b)

The same language can be represented in GNF as:

$$\begin{aligned}&S \to aAB \mid aB \mid bB \mid a\\&A \to aA \mid a \mid b\\&B \to b\end{aligned}$$

15.1 CFG to GNF

There are 3 major steps involved in the conversion of CFG to Greibach Normal Form:

1. **Remove Unit and Null Productions:**

   Check if the given CFG has any Unit Productions or Null Productions and remove them if there are any (using the Unit & Null Productions removal techniques).

   Formally:

   (a) Remove $A \to \varepsilon$ (Null productions)
   (b) Remove $A \to B$ where $A, B \in V$ (Unit productions)

2. **Convert to Chomsky Normal Form (CNF):**

   Check whether the CFG is already in Chomsky Normal Form and if not, then convert it to CNF.

   Formally: Ensure all productions are of the form:

   $$A \to BC \quad \text{or} \quad A \to a$$

   where $A, B, C \in V$ (non-terminals) and $a \in \Sigma$ (terminal).

3. **Convert CNF to GNF:**

   a) **Rename non-terminals:**
   Change the names of the non-terminal symbols to $A_1, A_2, \ldots, A_n$ in ascending order.
   Formally: Let $V = \{A_1, A_2, \ldots, A_n\}$ where A_1 is the start symbol.

   b) **Order the productions:**
   Alter the rules so that the non-terminals are in ascending order. For any production of the form:

   $$A_i \to A_j\alpha$$

   where $\alpha \in (V \cup \Sigma)^*$, we must ensure that:

   $$i < j$$

   If $i \geq j$: Replace A_j with its production rules to eliminate the violation.
   Formally: If $A_j \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_k$, then substitute:

   $$A_i \to \beta_1\alpha \mid \beta_2\alpha \mid \cdots \mid \beta_k\alpha$$

   c) **Remove left recursion:**
   If after substitution, we have direct left recursion of the form:

   $$A_i \to A_i\alpha_1 \mid A_i\alpha_2 \mid \cdots \mid A_i\alpha_r \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_s$$

   where β_j does not start with A_i, then eliminate it using:

   $$\begin{aligned} A_i &\to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_s \mid \beta_1A_{n+1} \mid \beta_2A_{n+1} \mid \cdots \mid \beta_sA_{n+1} \\ A_{n+1} &\to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_r \mid \alpha_1A_{n+1} \mid \alpha_2A_{n+1} \mid \cdots \mid \alpha_rA_{n+1} \end{aligned}$$

   where A_{n+1} is a new non-terminal.

   d) **Convert to terminal-leading form:**
   For each production $A_i \to A_j\alpha$ where $i < j$, substitute A_j with its productions until all productions are of the form:

   $$A_i \to a\gamma$$

   where $a \in \Sigma$ (terminal) and $\gamma \in V^*$ (string of non-terminals).

Example:
Given

$$\begin{aligned} S &\to CA \mid BB \\ B &\to 0 \mid SB \\ C &\to 0 \\ A &\to 1 \end{aligned}$$

1. Step 1: Rename non-terminals to f_i form

   We change the non-terminals into a numbered sequence: $S \to f_1$, $C \to f_2$, $A \to f_3$, $B \to f_4$

$$\begin{aligned} f_1 &\to f_2 f_3 \mid f_4 f_4 \\ f_2 &\to 0 \\ f_3 &\to 1 \\ f_4 &\to 0 \mid f_1 f_4 \end{aligned}$$

2. Step 2: Check ordering rule $(i < j)$

   Analyze each production $f_i \to f_j \alpha$:

   (a) $f_1 \to f_2 f_3$: $1 < 2$
   (b) $f_1 \to f_4 f_4$: $1 < 4$
   (c) $f_4 \to f_1 f_4$: $4 \not< 1$ (violates rule)

   The production $f_4 \to f_1 f_4$ violates the ordering rule because $i = 4 \geq j = 1$.

3. Step 3: Substitute f_1 into f_4 to fix violation

   Since $f_1 \to f_2 f_3 \mid f_4 f_4$, substitute into $f_4 \to f_1 f_4$:

$$\begin{aligned} f_4 &\to 0 \mid \underline{f_1} f_4 \\ &\to 0 \mid \underline{f_2 f_3} f_4 \mid \underline{f_4 f_4} f_4 \\ &\to 0 \mid f_2 f_3 f_4 \mid f_4 f_4 f_4 \end{aligned}$$

   Now, since $f_2 \to 0$, replace f_2 by 0 where it appears:

   Thus

$$f_4 \to 0 \mid 0 f_3 f_4 \mid f_4 f_4 f_4$$

4. Step 4: Remove left recursion from f_4

   The production $f_4 \to f_4 f_4 f_4$ has direct left recursion. Using the formula:

$$A \to A\alpha \mid \beta \quad \to \quad A \to \beta \mid \beta A', \quad A' \to \alpha \mid \alpha A'$$

   Here: $\alpha = f_4 f_4$ and $\beta = 0 \mid 0 f_3 f_4$

   Introduce new variable Δ:

$$\begin{aligned} &\Delta \to f_4 f_4 \mid f_4 f_4 \Delta \\ &f_4 \to 0 \mid 0 f_3 f_4 \mid 0\Delta \mid 0 f_3 f_4 \Delta \end{aligned}$$

5. Step 5: Convert all productions to GNF

   Current grammar:

$$\begin{aligned} &f_1 \to f_2 f_3 \mid f_4 f_4 \\ &f_2 \to 0 \\ &f_3 \to 1 \\ &f_4 \to 0 \mid 0 f_3 f_4 \mid 0\Delta \mid 0 f_3 f_4 \Delta \\ &\Delta \to f_4 f_4 \mid f_4 f_4 \Delta \end{aligned}$$

   Substitute $f_2 \to 0$ only in f_1 :

$$f_1 \to 0 f_3 \mid f_4 f_4$$

   Substitute f_4 into $f_1 \to f_4 f_4$. Since $f_4 \to 0 \mid 0 f_3 f_4 \mid 0\Delta \mid 0 f_3 f_4 \Delta$:

$$f_1 \to 0 f_3 \mid 0 f_4 \mid 0 f_3 f_4 f_4 \mid 0\Delta f_4 \mid 0 f_3 f_4 \Delta f_4$$

   Substitute f_4 into Δ:

$$\begin{aligned} \Delta \to\ & 0 f_4 \mid 0 f_3 f_4 f_4 \mid 0\Delta f_4 \mid 0 f_3 f_4 \Delta f_4 \\ & \mid 0 f_4 \Delta \mid 0 f_3 f_4 f_4 \Delta \mid 0\Delta f_4 \Delta \mid 0 f_3 f_4 \Delta f_4 \Delta \end{aligned}$$

6. Step 6: Final Grammar in GNF

$$\begin{aligned} &f_1 \to 0 f_3 \mid 0 f_4 \mid 0 f_3 f_4 f_4 \mid 0\Delta f_4 \mid 0 f_3 f_4 \Delta f_4 \\ &f_2 \to 0 \\ &f_3 \to 1 \\ &f_4 \to 0 \mid 0 f_3 f_4 \mid 0\Delta \mid 0 f_3 f_4 \Delta \\ &\Delta \to 0 f_4 \mid 0 f_3 f_4 f_4 \mid 0\Delta f_4 \mid 0 f_3 f_4 \Delta f_4 \\ &\qquad \mid 0 f_4 \Delta \mid 0 f_3 f_4 f_4 \Delta \mid 0\Delta f_4 \Delta \mid 0 f_3 f_4 \Delta f_4 \Delta \end{aligned}$$

   Hence, all productions start with a terminal symbol (0 or 1) followed by zero or more non-terminals.

16 Pushdown Automata (PDA)

A Pushdown Automaton (PDA) is formally defined as a 7-tuple:

$$M = (W, \Sigma, \Gamma, V, S, \Delta, F)$$

where:

1. W is a finite set of states
2. Σ is a finite input alphabet
3. Γ is a finite stack alphabet
4. $V : W \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}(W \times \Gamma^*)$ is the transition function
5. $S \in W$ is the initial state
6. $\Delta \in \Gamma$ is the initial stack symbol
7. $F \subseteq W$ is the set of accepting/final states

16.1 Transition Function

The transition function $V(w, a, X)$ returns a set of pairs (w', γ) where:

1. w is the current state
2. $a \in \Sigma \cup \{\varepsilon\}$ is the input symbol (or ε for no input)
3. $X \in \Gamma$ is the top symbol of the stack
4. w' is the next state
5. $\gamma \in \Gamma^*$ is the string to replace X on the stack

Example:
Given

$$L = \{0^n 1^n \mid n \geq 0\}$$

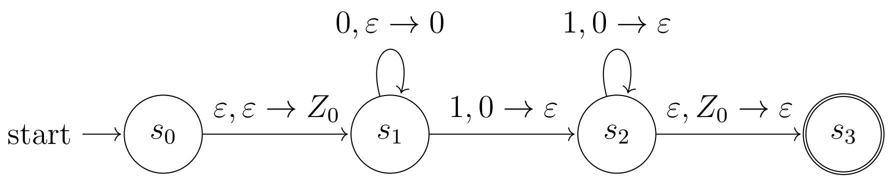
We design a PDA that accepts strings with equal numbers of 0's followed by 1's.

1. PDA Specification

$$\begin{aligned} W &= \{s_0, s_1, s_2, s_3\} & S &= s_0 \\ \Sigma &= \{0, 1\} & Z_0 &= Z_0 \\ \Gamma &= \{Z_0, 0\} & F &= \{s_3\} \end{aligned}$$

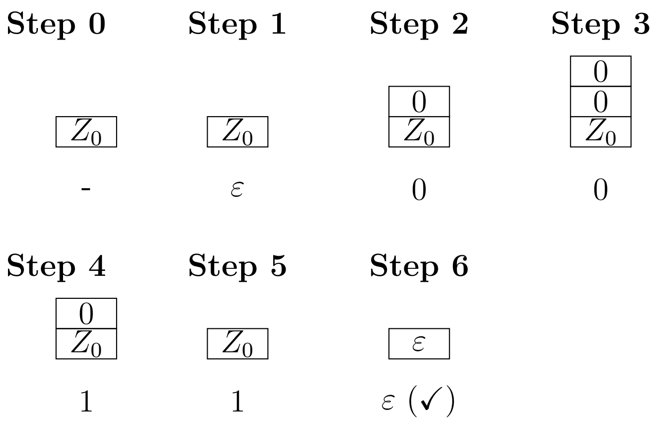
2. Transition Rules

$V(s_0, \varepsilon, \varepsilon) = \{(s_1, Z_0)\}$	Initialize stack with Z_0
$V(s_1, 0, \varepsilon) = \{(s_1, 0)\}$	Push 0 for each 0
$V(s_1, 1, 0) = \{(s_2, \varepsilon)\}$	Pop 0 for first 1
$V(s_2, 1, 0) = \{(s_2, \varepsilon)\}$	Pop 0 for each 1
$V(s_2, \varepsilon, Z_0) = \{(s_3, \varepsilon)\}$	Accept if all 0s matched



Execution for 0011

Step	**State**	**Input Symbol Read**	**Stack**	**Action**
0	s_0	-	Z_0	Initial configuration
1	s_1	ε	Z_0	ε-transition, push Z_0
2	s_1	0	$0Z_0$	Read 0, push 0
3	s_1	0	$00Z_0$	Read 0, push 0
4	s_2	1	$0Z_0$	Read 1, pop 0
5	s_2	1	Z_0	Read 1, pop 0
6	s_3	ε	ε	ε-transition, accept



The string 0011 is accepted because we end in an accepting state with the input fully consumed.

16.2 CFG to PDA

Given CFG $G = (N, \Sigma, R, S)$, construct PDA $M = (W, \Sigma, \Gamma, V, S, Z_0, F)$ where:

$$W = \{q\} \qquad \Gamma = N \cup \Sigma \qquad Z_0 = Z_0 \qquad F = \{q\}$$

Transition function V:

$$\begin{aligned} V(q, \varepsilon, Z_0) &= \{(q, S)\} && \text{Initialize stack with start symbol} \\ V(q, \varepsilon, A) &= \{(q, \beta) \mid A \to \beta \in R\} && \text{Apply production rules for } A \in N \\ V(q, a, a) &= \{(q, \varepsilon)\} && \text{Match terminals for } a \in \Sigma \end{aligned}$$

Example:
Given CFG

$$S \to 0S1 \mid \varepsilon$$

This generates $L = \{0^n 1^n \mid n \geq 0\}$

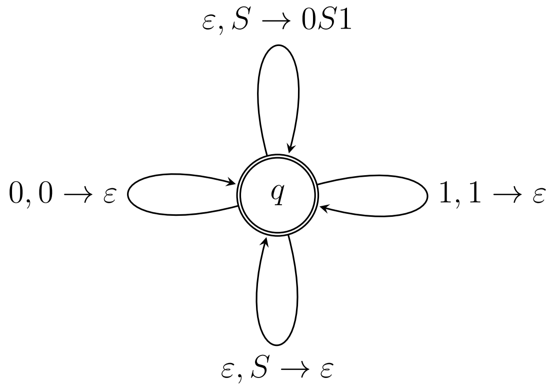
1. PDA Specification

$$\begin{aligned} W &= \{q\} & S &= q \\ \Sigma &= \{0, 1\} & Z_0 &= Z_0 \\ \Gamma &= \{Z_0, S, 0, 1\} & F &= \{q\} \end{aligned}$$

2. Transition Rules

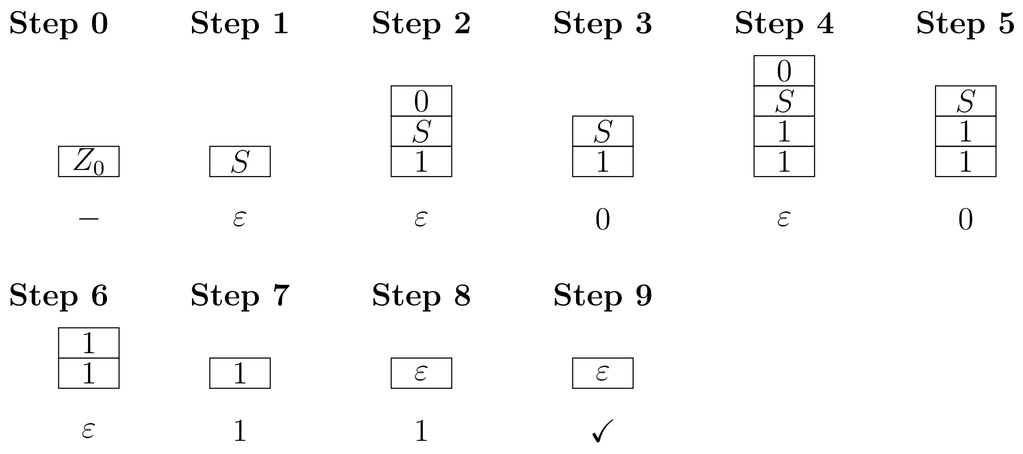
$$\begin{aligned} V(q, \varepsilon, Z_0) &= \{(q, S)\} && \text{Initialize stack with } S \\ V(q, \varepsilon, S) &= \{(q, 0S1), (q, \varepsilon)\} && \text{Apply } S \to 0S1 \text{ or } S \to \varepsilon \\ V(q, 0, 0) &= \{(q, \varepsilon)\} && \text{Match and pop 0} \\ V(q, 1, 1) &= \{(q, \varepsilon)\} && \text{Match and pop 1} \end{aligned}$$

3. PDA Transition Diagram:



Execution for $w = 0011$

Step	**State**	**Input Symbol Read**	**Stack**	**Action**
0	q	-	Z_0	Initial configuration
1	q	ε	S	ε-transition, push S
2	q	ε	$0S1$	Apply $S \to 0S1$
3	q	0	$S1$	Read 0, pop 0
4	q	ε	$0S11$	Apply $S \to 0S1$
5	q	0	$S11$	Read 0, pop 0
6	q	ε	11	Apply $S \to \varepsilon$
7	q	1	1	Read 1, pop 1
8	q	1	ε	Read 1, pop 1
9	q	ε	ε	Accept



Stack is shown with top at the top. Input remaining shown below each step. The PDA accepts by empty stack.

16.3 PDA to CFG

Given PDA $M = (W, \Sigma, \Gamma, V, w_0, Z_0, F)$, construct CFG $G = (N, \Sigma, R, S)$ where:

1. Non-terminals: $N = \{S\} \cup \{[w, A, w'] \mid w, w' \in W, A \in \Gamma\}$

   Each non-terminal $[w, A, w']$ represents: "starting from state w with A on top of stack, reach state w' with A popped"

2. Start symbol: S with production:

$$S \to [w_0, Z_0, w_f] \quad \forall w_f \in F$$

3. Production rules:

   (a) For each transition $(w', \varepsilon) \in V(w, a, A)$ (pop):

$$[w, A, w'] \to a \quad \text{where } a \in \Sigma \cup \{\varepsilon\}$$

   (b) For each transition $(w', B_1B_2...B_k) \in V(w, a, A)$ with $k \geq 1$ (push $B_1B_2...B_k$):

$$[w, A, w_{k+1}] \to a[w', B_1, w_1][w_1, B_2, w_2]...[w_k, B_k, w_{k+1}]$$

   for all possible intermediate states $w_1, w_2, ..., w_k \in W$

   (c) For each state $w \in W$ and stack symbol $A \in \Gamma$:

$$[w, A, w] \to \varepsilon$$

Example:
Given PDA for $L = \{0^n1^n \mid n \geq 0\}$:

1. PDA Specification

$$\begin{aligned} W &= \{w_0, w_1, w_2, w_3\} & w_0 &= w_0 \\ \Sigma &= \{0, 1\} & Z_0 &= Z_0 \\ \Gamma &= \{Z_0, 0\} & F &= \{w_3\} \end{aligned}$$

2. Transition Rules

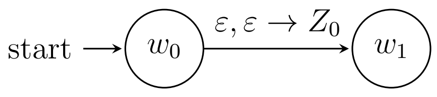
$$\begin{aligned} V(w_0, \varepsilon, \varepsilon) &= \{(w_1, Z_0)\} && \text{Initialize stack} \\ V(w_1, 0, Z_0) &= \{(w_1, 0Z_0)\} && \text{Push 0 on } Z_0 \\ V(w_1, 0, 0) &= \{(w_1, 00)\} && \text{Push 0 on 0} \\ V(w_1, \varepsilon, 0) &= \{(w_2, 0)\} && \text{Transition to pop phase} \\ V(w_2, 1, 0) &= \{(w_2, \varepsilon)\} && \text{Pop 0 for each 1} \\ V(w_2, \varepsilon, Z_0) &= \{(w_3, \varepsilon)\} && \text{Accept} \end{aligned}$$

Execution for $w = 0011$

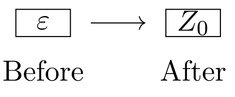
Step	**State**	**Input Symbol**	**Stack**	**Action**
0	w_0	-	ε	Initial configuration
1	w_1	ε	Z_0	Push Z_0
2	w_1	0	$0Z_0$	Read 0, push 0 on Z_0
3	w_1	0	$00Z_0$	Read 0, push 0 on 0
4	w_2	ε	$00Z_0$	Transition to pop phase
5	w_2	1	$0Z_0$	Read 1, pop 0
6	w_2	1	Z_0	Read 1, pop 0
7	w_3	ε	ε	Accept

3. Constructed CFG:

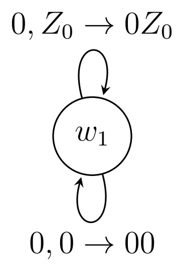
   (a) Initialization transition
   From
   $$V(w_0, \varepsilon, \varepsilon) = \{(w_1, Z_0)\},$$
   the PDA moves into w_1 and initializes the stack with Z_0.
   $$S \to [w_0, Z_0, w_3], \qquad [w_0, Z_0, w_3] \to [w_1, Z_0, w_3].$$



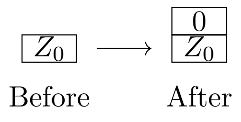
State: $w_0 \to w_1$



   (b) Push transitions
   From
   $$V(w_1, 0, Z_0) = \{(w_1, 0Z_0)\}, \qquad V(w_1, 0, 0) = \{(w_1, 00)\}.$$
   For all possible final states $w_i \in W$:
   $$[w_1, Z_0, w_i] \to 0[w_1, 0, w_i], \qquad [w_1, 0, w_i] \to 0[w_1, 0, w_i].$$



State: $w_1 \to w_1$, Read: 0



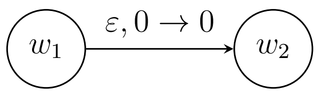
(c) Phase change: $w_1 \to w_2$

From

$$V(w_1, \varepsilon, 0) = \{(w_2, 0)\}.$$

This is a transition that doesn't pop (pushes the same symbol back):

$$[w_1, 0, w_i] \to [w_2, 0, w_i] \quad \text{for all } w_i \in W.$$



State: $w_1 \to w_2$

$$\boxed{0} \longrightarrow \boxed{0}$$

Before After

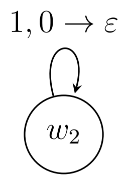
(d) Pop transition

From

$$V(w_2, 1, 0) = \{(w_2, \varepsilon)\}.$$

This pops the 0, so:

$$[w_2, 0, w_2] \to 1.$$



State: $w_2 \to w_2$, Read: 1

$$\boxed{0} \longrightarrow \boxed{\varepsilon}$$

Before After

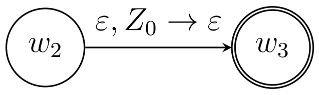
(e) Accept transition

From

$$V(w_2, \varepsilon, Z_0) = \{(w_3, \varepsilon)\}.$$

This pops Z_0:

$$[w_2, Z_0, w_3] \to \varepsilon.$$



State: $w_2 \to w_3$

$$\boxed{Z_0} \longrightarrow \boxed{\varepsilon}$$

Before After

The generated language:

$$L = \{0^n 1^n \mid n \geq 0\}.$$

$$\begin{tikzcd}
\text{start} \arrow[r] & w_0 \arrow[r, "{\varepsilon, \varepsilon \to Z_0}"] & w_1 \arrow[loop above, "{\begin{array}{c}0, Z_0 \to 0Z_0 \\ 0, 0 \to 00\end{array}}"] \arrow[r, "{\varepsilon, 0 \to 0}"] & w_2 \arrow[loop above, "{1, 0 \to \varepsilon}"] \arrow[r, "{\varepsilon, Z_0 \to \varepsilon}"] & \boxed{w_3}
\end{tikzcd}$$

Derivation Example for 0011:

$$\begin{aligned}
S &\to [w_0, Z_0, w_3] \\
&\to [w_1, Z_0, w_3] \\
&\to 0[w_1, 0, w_3] \\
&\to 00[w_1, 0, w_3] \\
&\to 00[w_2, 0, w_2][w_2, Z_0, w_3] \\
&\to 001[w_2, Z_0, w_3] \\
&\to 0011.
\end{aligned}$$

17 Palindromes

Formally

Let $\Sigma = \{0, 1\}$ be a binary alphabet and $w \in \Sigma^*$ be a string over Σ. We denote by w^R the reverse of w. Then w is a *palindrome* if and only if:

$$w = w^R$$

Equivalently, for a string $w = x_1 x_2 \cdots x_n$ where $x_i \in \Sigma$ and $n = |w|$, w is a palindrome if and only if:

$$x_i = x_{n-i+1} \quad \forall i \in \{1, 2, \ldots, n\}$$

In general:
Consider the string $w = 110011$ over $\Sigma = \{0, 1\}$.
We have:

$$w = 110011, \quad w^R = 110011$$

Since $w = w^R$, the string 110011 is a palindrome. Alternatively, checking each position:

$$\begin{aligned}
x_1 &= x_6 = 1 \\
x_2 &= x_5 = 1 \\
x_3 &= x_4 = 0
\end{aligned}$$

Therefore, 110011 is a palindrome.

Example:
Given

$$L = \{xx^R \mid x \in (0+1)^+\}$$

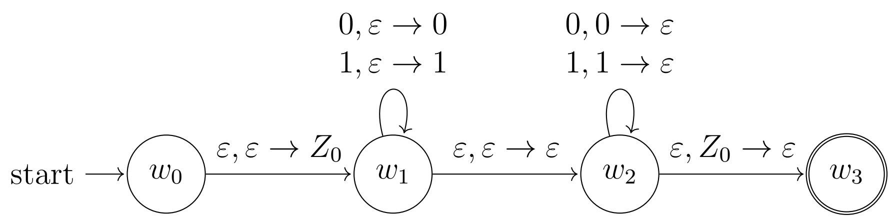
For input string $w = 0110$, we need to verify if it belongs to L.

1. PDA Specification

$$\begin{aligned} W &= \{w_0, w_1, w_2, w_3\} & S &= w_0 \\ \Sigma &= \{0, 1\} & Z_0 &= Z_0 \\ \Gamma &= \{Z_0, 0, 1\} & F &= \{w_3\} \end{aligned}$$

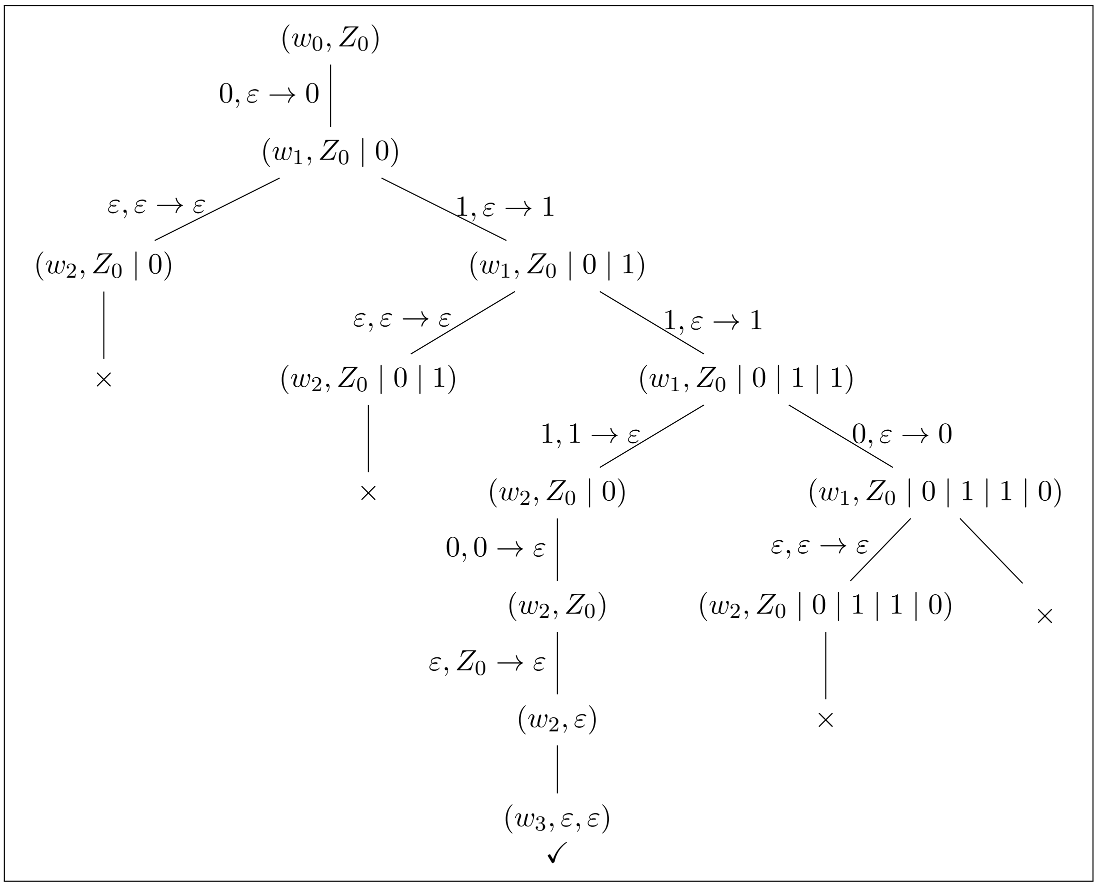
2. Transition Rules

$V(w_0, \varepsilon, \varepsilon) = \{(w_1, Z_0)\}$	Initialize stack with Z_0
$V(w_1, 0, \varepsilon) = \{(w_1, 0)\}$	Push 0 onto stack
$V(w_1, 1, \varepsilon) = \{(w_1, 1)\}$	Push 1 onto stack
$V(w_1, \varepsilon, \varepsilon) = \{(w_2, \varepsilon)\}$	Guess middle, transition to pop phase
$V(w_2, 0, 0) = \{(w_2, \varepsilon)\}$	Match and pop 0
$V(w_2, 1, 1) = \{(w_2, \varepsilon)\}$	Match and pop 1
$V(w_2, \varepsilon, Z_0) = \{(w_3, \varepsilon)\}$	Accept if stack is empty



3. Computation Tree for Input 0110

   Since the PDA is nondeterministic (can guess the middle at any point using ε-transition from w_1 to w_2), we explore all possible computation paths with this schema $[\varepsilon \mid 0 \mid \varepsilon \mid 1 \mid \varepsilon \mid 1 \mid \varepsilon \mid 0 \mid \varepsilon]$


(w_0, Z_0)
$0, \varepsilon \to 0$
$(w_1, Z_0 \mid 0)$
$\varepsilon, \varepsilon \to \varepsilon$
$1, \varepsilon \to 1$
$(w_2, Z_0 \mid 0)$
$(w_1, Z_0 \mid 0 \mid 1)$
$\times$
$\varepsilon, \varepsilon \to \varepsilon$
$1, \varepsilon \to 1$
$(w_2, Z_0 \mid 0 \mid 1)$
$(w_1, Z_0 \mid 0 \mid 1 \mid 1)$
$\times$
$1, 1 \to \varepsilon$
$0, \varepsilon \to 0$
$(w_2, Z_0 \mid 0)$
$(w_1, Z_0 \mid 0 \mid 1 \mid 1 \mid 0)$
$0, 0 \to \varepsilon$
$\varepsilon, \varepsilon \to \varepsilon$
(w_2, Z_0)
$(w_2, Z_0 \mid 0 \mid 1 \mid 1 \mid 0)$
$\times$
$\varepsilon, Z_0 \to \varepsilon$
(w_2, ε)
$\times$
$(w_3, \varepsilon, \varepsilon)$
✓

(a) The PDA must nondeterministically guess where the middle of the string is

(b) The accepting path: Push 0, push 1, guess middle (via ε-transition), pop 1, pop 0

(c) This corresponds to recognizing 01|10 where | marks the middle

(d) Other guesses of the middle position lead to rejection

Step	**State**	**Remaining Input**	**Stack**	**Action**
0	w_0	-	ε	Initial configuration
1	w_1	Z_0	Z_0	ε-transition, push Z_0
2	w_1	$0Z_0$	$0Z_0$	Read 0, push 0
3	w_1	10	$10Z_0$	Read 1, push 1
4	w_2	10	$10Z_0$	ε-transition (guess middle)
5	w_2	0	$0Z_0$	Read 1, match and pop 1
6	w_2	ε	Z_0	Read 0, match and pop 0
7	w_3	ε	ε	ε-transition, accept

4. Concisely

   For input

$$w = 0110 \in L = \{xx^R \mid x \in \{0,1\}^+\}$$

   (a) String Decomposition

$$w = 0110 = \underbrace{01}_{x}\underbrace{10}_{x^R} \quad \text{where } |x| = 2$$

(b) Meeting Point Function

$$\text{MeetPoint}(w) = n = 2 \quad \text{such that } w[1..n] = x \text{ and } w[n+1..2n] = x^R$$

(c) Configuration at Middle (Step 4)
Using stack notation $Z_0 a_1 a_2 \dots a_k$ where Z_0 is leftmost (bottom) and elements grow rightward (top on right):

$$(w_1, Z_0 \mid 0 \mid 1) \xrightarrow{\varepsilon,\varepsilon\to\varepsilon} (w_2, Z_0 \mid 0 \mid 1)$$

This is the critical nondeterministic transition where the PDA "guesses" the middle.

(d) Invariant at Meeting Point

$$\boxed{\text{Stack}(w_2) = Z_0 01 \quad \wedge \quad \text{RemainingInput}(w_2) = 10 = x^R}$$

This captures the symmetry: what remains to read (10) equals the reverse of what's been stored (01).

(e) Push-Meet-Pop Schema

$$\underbrace{w[1..2] = 01}_{\text{read \& push}} \xrightarrow[\text{Steps 2-3}]{\text{push}} \underbrace{\text{Stack} = Z_0 01}_{\text{Step 4}} \xrightarrow[\text{Steps 5-6}]{\text{match \& pop}} \underbrace{w[3..4] = 10}_{\text{verify } x^R}$$

(f) Formal Acceptance Condition

$(w_0, Z_0) \vdash (w_1, Z_0 \mid 0) \vdash (w_1, Z_0 \mid 0 \mid 1) \vdash (w_2, Z_0 \mid 0 \mid 1) \vdash (w_2, Z_0 \mid 0) \vdash (w_2, Z_0)$
$\vdash (w_2, \varepsilon) \vdash (w_3, \varepsilon)$

18 Turing Machine

A TM is formally defined as a 7-tuple:

$$M = (W, \Sigma, \Gamma, V, S, B, F)$$

where:

1. W is a finite set of states
2. Σ is the finite input alphabet not containing the blank symbol B
3. Γ is the finite tape alphabet, where $B \in \Gamma \wedge \Sigma \subseteq \Gamma$
4. $V : W \times \Gamma \to W \times \Gamma \times \{L, R\}$ is the transition function
5. $S \in W$ is the start state
6. $B \in \Gamma$ is the blank symbol
7. $F \subseteq W$ is the set of final (accepting) states

Example 1:
Given

$$L = \{0^n 1^n \mid n \geq 1\}$$

Let $M = (W, \Sigma, \Gamma, V, S, B, F)$ where:

$$W = \{w_0, w_1, w_2, w_3\} \qquad \Sigma = \{0, 1\} \qquad \Gamma = \{0, 1, X, Y, B\}$$

$$S = w_0 \qquad B = \text{␣} \qquad F = \{w_3\}$$

Transition Function V

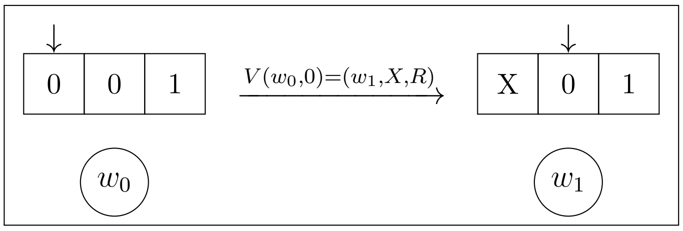
$V(w_0, 0) = (w_1, X, R)$	Mark first 0, move right
$V(w_0, Y) = (w_0, Y, R)$	Skip marked 1s
$V(w_0, B) = (w_3, B, R)$	Accept if tape is blank
$V(w_1, 0) = (w_1, 0, R)$	Scan right past 0s
$V(w_1, Y) = (w_1, Y, R)$	Scan right past marked 1s
$V(w_1, 1) = (w_2, Y, L)$	Mark first 1, move left
$V(w_2, 0) = (w_2, 0, L)$	Scan left past 0s
$V(w_2, Y) = (w_2, Y, L)$	Scan left past marked 1s
$V(w_2, X) = (w_0, X, R)$	Return to marked 0, repeat

All other transitions are undefined (reject).

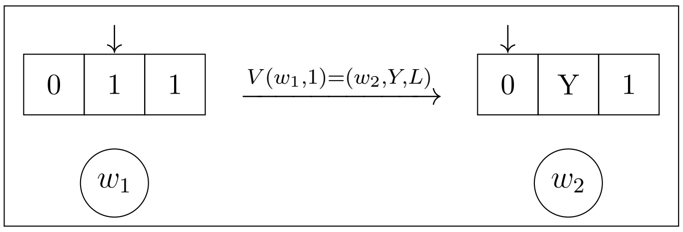
Example Execution on Input “0011”

Turing Machine Transitions

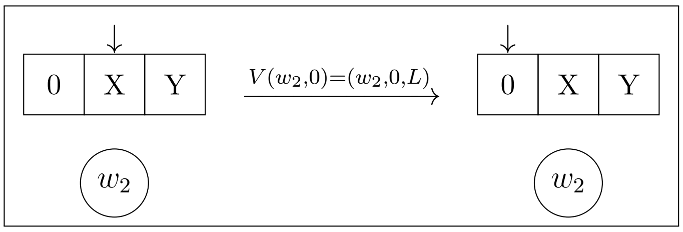
In state w_0 the machine reads 0, writes X, moves right, and enters state w_1.



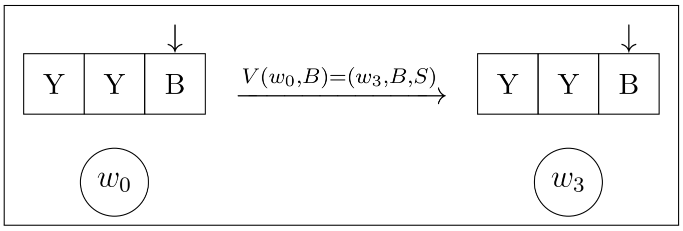
In state w_1 the machine reads 1, writes Y, moves left, and enters state w_2.



In state w_2 the machine reads 0, writes 0, moves left, and stays in w_2.



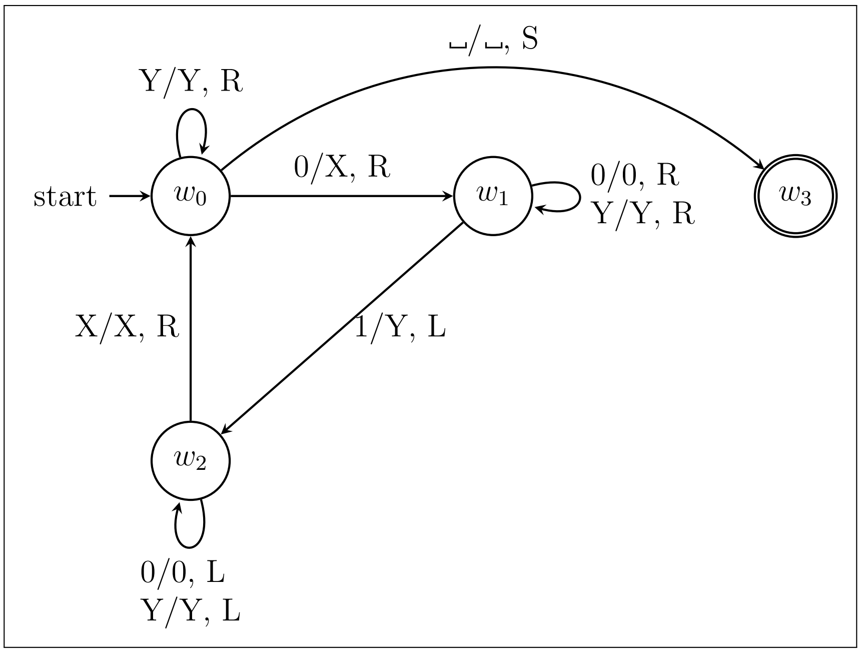
In state w_0 the machine reads B, writes B, does not move, and enters the accepting state w_3.



Example 2:
Given

$$T = \{0^k1^k \mid k \geq 1\}$$

with the TM defined as:



where the components are defined as:

$$W = \{w_0, w_1, w_2, w_3\} \qquad \Sigma = \{0, 1\} \qquad \Gamma = \{0, 1, X, Y, ␣\}$$

$$S = w_0 \qquad B = ␣ \qquad F = \{w_3\}$$

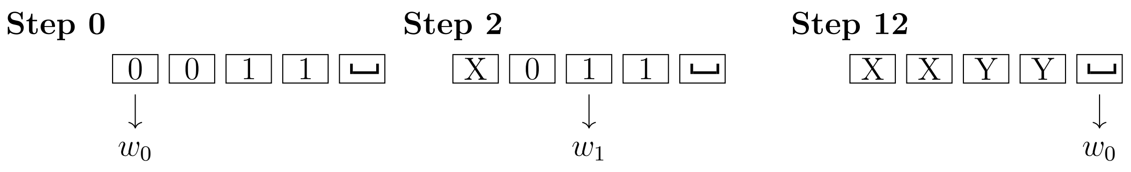
1. Execution Trace for Input "0011"

Step	**State**	**Tape Content**	**Head Position**	**Action**
0	w_0	$\boxed{0}011$	0	Read 0, write X, move R
1	w_1	$X\boxed{0}11$	1	Read 0, skip, move R
2	w_1	$X0\boxed{1}1$	2	Read 1, write Y, move L
3	w_2	$X\boxed{0}Y1$	1	Read 0, skip, move L
4	w_2	$\boxed{X}0Y1$	0	Read X, move R
5	w_0	$X\boxed{0}Y1$	1	Read 0, write X, move R
6	w_1	$XX\boxed{Y}1$	2	Read Y, skip, move R
7	w_1	$XXY\boxed{1}$	3	Read 1, write Y, move L
8	w_2	$XX\boxed{Y}Y$	2	Read Y, skip, move L
9	w_2	$X\boxed{X}YY$	1	Read X, move R
10	w_0	$XX\boxed{Y}Y$	2	Read Y, skip, move R
11	w_0	$XXY\boxed{Y}$	3	Read Y, skip, move R
12	w_0	$XXYY\boxed{␣}$	4	Read ␣, accept
13	w_3	XXYY	4	**ACCEPT**

2. Transition Table

Current State	**Read Symbol**	**Write Symbol**	**Next State, Direction**
w_0	0	X	(w_1, R)
w_0	Y	Y	(w_0, R)
w_0	␣	␣	(w_3, S)
w_1	0	0	(w_1, R)
w_1	Y	Y	(w_1, R)
w_1	1	Y	(w_2, L)
w_2	0	0	(w_2, L)
w_2	Y	Y	(w_2, L)
w_2	X	X	(w_0, R)
w_3	-	-	Accept

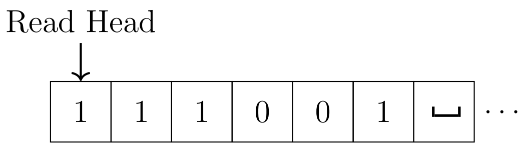
Illustratively,



18.1 Turing Machine Programming Techniques

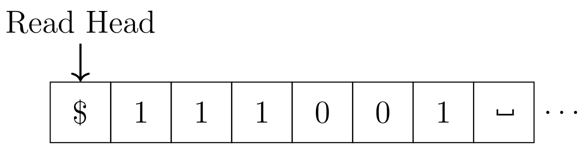
A standard TM cannot directly detect the left end of its tape. Hence, we place a special symbol \$ at the left end of the tape and shift the input one cell to the right.

Example:



1. Step 1: Shift input one cell to the right and place \$ at position 0

   The Turing machine reads the input tape and shifts all symbols one position to the right, then places the special symbol \$ at the leftmost position. This allows the machine to detect when it reaches the left boundary.



2. Step 2: Left boundary detection

   When the Turing machine reads the \$ symbol, it knows it has reached the leftmost position and cannot move further left. This prevents the machine from falling off the tape.

Scenario	**Read Symbol**	**Action**
Attempt to move left past \$	\$	Reject or stay in place
Normal left movement	0, 1, or ␣	Move left normally
Detect left boundary	\$	Signal end of leftward scanning

Example Execution:

(a) Initial Configuration

$$\boxed{\$}\;\boxed{1}\;\boxed{0}\;\boxed{1}$$

$$\downarrow$$

Scanning left

(b) Move left from position 1

$$\boxed{\$}\;\boxed{1}\;\boxed{0}\;\boxed{1}$$

$$\downarrow$$

Read \$

(c) Detection: Reached left boundary

$$\boxed{\text{Machine recognizes \$ and stops attempting leftward movement}}$$

18.2 Comparing Two Strings

To compare two strings separated by # using a Turing machine:

1. Mark leftmost unmarked symbol in first string ($0 \to$ X, $1 \to$ Y)
2. Scan right past # to leftmost unmarked symbol in second string
3. If symbols match, mark it; otherwise reject
4. Return to left end
5. Repeat until first string exhausted
6. Accept iff second string also exhausted; otherwise reject

Example:
Given

$$L = \{w\#w \mid w \in \{0,1\}^*\}$$

1. Accept 01#01

$$\begin{aligned} 01\#01 &\to x1\#x1 \quad \text{(mark first 0s)} \\ &\to xy\#xy \quad \text{(mark first 1s)} \\ &\to \text{Accept} \end{aligned}$$

2. Reject 10#01

$$\begin{aligned}10\#01 &\rightarrow y0\#01 \quad \text{(mark first symbol: 1)}\\ &\rightarrow \text{Compare with 0: mismatch}\\ &\rightarrow \text{reject}\end{aligned}$$

3. Reject 01#011 (unequal length)

$$\begin{aligned}01\#011 &\rightarrow x1\#x11 \quad \text{(mark 0s)}\\ &\rightarrow xy\#xy1 \quad \text{(mark 1s)}\\ &\rightarrow \text{First string exhausted but unmarked 1 remains}\\ &\rightarrow \text{reject}\end{aligned}$$

18.3 Multiple Turing Machines

A multi-tape Turing machine has multiple tapes, each with its own read/write head. The transition function is defined as:

$$V : W \times \Gamma^k \rightarrow W \times \Gamma^k \times \{L, R, S\}^k$$

where k is the number of tapes, W is the set of states, and Γ is the tape alphabet.

Example:

A multi-tape Turing machine has multiple tapes, each with its own read/write head. The transition function is defined as:

$$V : W \times \Gamma^k \rightarrow W \times \Gamma^k \times \{L, R, S\}^k$$

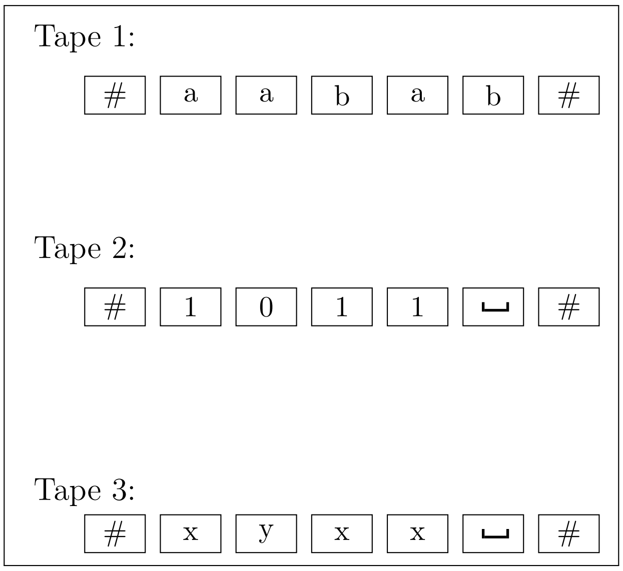
where k is the number of tapes, W is the set of states, and Γ is the tape alphabet.

Example:

Consider a 3-tape Turing machine simulating a finite state machine. The single tape contains the entire configuration encoded across three logical tracks:

1. Tape 1: Contains the input string (e.g., #aabab#)
2. Tape 2: Contains working data (e.g., #1011#)
3. Tape 3: Contains auxiliary symbols (e.g., #xyxx#)

The machine uses special markers (like #) to delimit sections and can simulate the FSM by coordinating operations across all three tapes simultaneously. Each tape has its own read/write head that can move independently according to the transition function V.



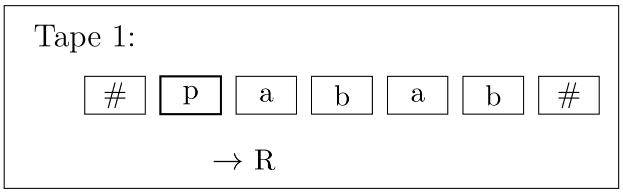
Suppose the TM is in state w_i and reads:

1. Tape 1: a
2. Tape 2: 1
3. Tape 3: x

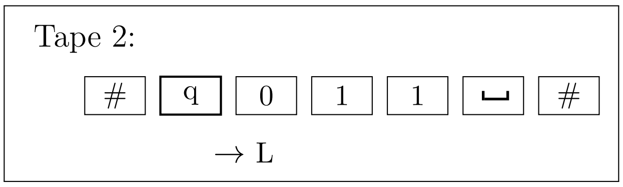
The transition function processes all three symbols simultaneously:

$$V(w_i, (a, 1, x)) = (w_j, (p, q, r), (R, L, R))$$

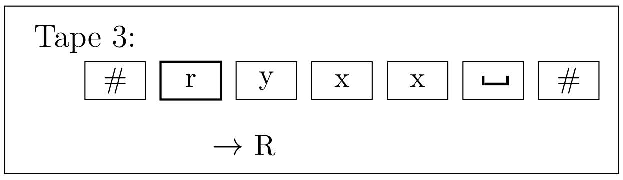
1. Write p on Tape 1, move Head to the right



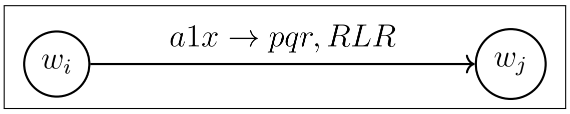
2. Write q on Tape 2, move Head to the left



3. Write r on Tape 3, move Head to the right



4. Transition to state w_j



18.4 Nondeterministic Turing Machine

For a nondeterministic Turing machine (NTM), the transition function is defined as:

$$V : W \times \Gamma \to \mathcal{P}(W \times \Gamma \times \{L, R\})$$

This means that for a given state and tape symbol, the NTM can transition to a set of possible states, write a new symbol, and move the head in either direction.

Example:

Let:

$$W = \{q_0, q_1, q_2\}, \quad \Gamma = \{0, 1, \sqcup\}$$

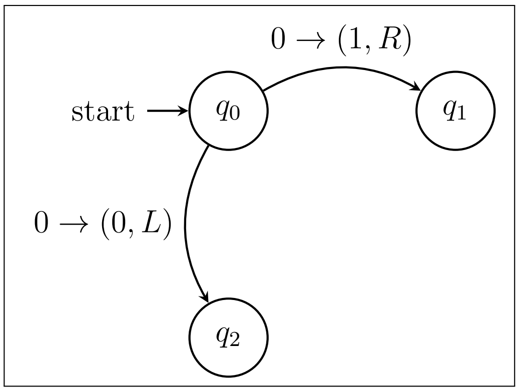
Suppose the machine is in state q_0 and reads the symbol 0. The transition function is defined as:

$$V(q_0, 0) = \{(q_1, 1, R),\ (q_2, 0, L)\}$$

This means that the NTM has two possible transitions:

1. Write 1, move the head to the right, and enter state q_1.
2. Leave the symbol unchanged, move the head to the left, and enter state q_2.

The computation branches nondeterministically, and the machine accepts if at least one computation path reaches an accepting state.



18.5 The Equivalence of Nondeterministic Turing Machine and Deterministic Turing Machine

An NTM can be simulated by a deterministic Turing machine (DTM) by systematically exploring all possible computation paths of the NTM. The deterministic machine performs a *breadth-first search* (BFS) over the computation tree of the NTM, ensuring that if any branch leads to an accepting state, the DTM will eventually find it.

Example

Consider an NTM $M_N = (W, \Gamma, V, q_0, q_{\text{acc}}, q_{\text{rej}})$ defined as follows:

$$W = \{q_0, q_1, q_2, q_{\text{acc}}\}, \quad \Gamma = \{0, 1, \sqcup\}$$

The transition function contains nondeterministic choices:

$$V(q_0, 0) = \{(q_1, 0, R),\ (q_2, 0, R)\}$$

$$V(q_1, \sqcup) = \{(q_{\text{acc}}, \sqcup, R)\}, \quad V(q_2, \sqcup) = \{(q_{\text{rej}}, \sqcup, R)\}$$

The NTM accepts an input string if at least one computation path reaches q_{acc}.

1. Computation Tree of the NTM

   For the input string 0, the NTM produces the following computation tree:

$$(q_0, 0) \ \rightarrow \ \begin{cases} (q_1, \sqcup) & \rightarrow \ q_{\text{acc}} \\ (q_2, \sqcup) & \rightarrow \ q_{\text{rej}} \end{cases}$$

   Since one branch leads to an accepting state, the NTM accepts the input.

2. DTM Simulation via Breadth-First Search

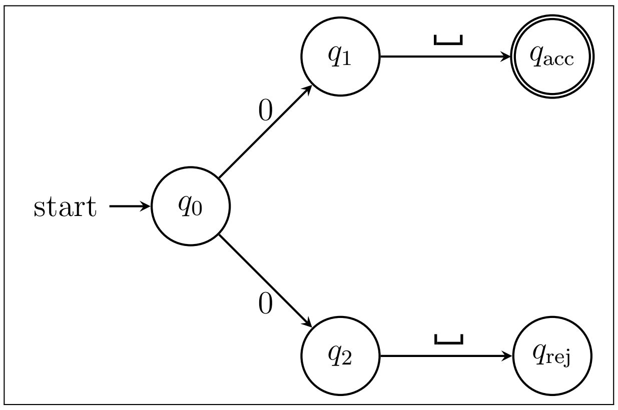
   A deterministic Turing machine M_D simulates M_N by enumerating all possible configurations of M_N level by level. At each step, M_D:

   (a) Stores configurations of M_N on its tape.

   (b) Expands all configurations at the current depth.

   (c) Checks whether any configuration is accepting.

   Because the simulation is breadth-first, M_D is guaranteed to discover an accepting configuration at finite depth if one exists.



18.6 Turing Machine as a Problem Solver

A TM can be designed to solve specific computational problems by defining its states, tape alphabet, and transition functions to manipulate input data according to the problem's requirements.

Example:

Let

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

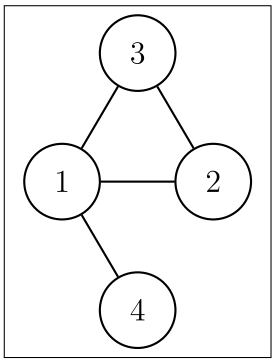
where the encoding of a graph G is given as a list of edges:

$$\langle G \rangle = (\{1,2\},\{1,3\},\{1,4\},\{2,3\})$$

and the input alphabet is

$$\Sigma = \{(,),\{,\},,,0,1,2,3,4,\ldots\}$$

The graph G consists of vertices $\{1,2,3,4\}$ and edges connecting vertex pairs as specified above.



The input graph is encoded on the TM tape as a linear string using symbols from Σ. The encoding is:

$$(\{1,2\},\{1,3\},\{1,4\},\{2,3\})$$

Initially, the tape contains the encoding followed by blank symbols, with the read/write head positioned at the first cell:

(	{	1	,	2	}	,	{	1	,	3	}	,	{	1	,	4	}	,	{	2	,	3	}	)	␣

The TM processes the input to determine if the graph is connected by exploring the vertices and edges according to its transition functions. If it finds that all vertices are reachable from any starting vertex, it enters an accepting state; otherwise, it rejects the input.

18.7 The Universal Turing Machine

A Universal Turing Machine (UTM) is a theoretical construct that can simulate any other Turing machine. It takes as input a description of a Turing machine M and an input string x, and it simulates the behavior of M on x.

Example:

Let M be a Turing machine defined as follows:

1. States: $W = \{q_0, q_1, q_{\text{acc}}, q_{\text{rej}}\}$
2. Tape Alphabet: $\Gamma = \{0, 1, \sqcup\}$
3. Transition Function:

$$V(q_0, 0) = (q_1, 1, R), \quad V(q_1, \sqcup) = (q_{\text{acc}}, \sqcup, R)$$

The UTM U simulates M on input $x = 000$ as follows:

$$\begin{tikzcd}
\langle M, 000\rangle \arrow[r] & \boxed{\text{UTM } U} \arrow[r] & \text{Simulates } M \text{ on } 000
\end{tikzcd}$$

U reads the description of M and input 000, then simulates M's computation step-by-step

19 Decidability and Undecidability

1. Decidable Languages

   A language L is decidable if it is a recursive language. All decidable languages are recursive and vice versa.

   Example:

$$L = \{0^n 1^n \mid n \geq 0\}$$

   A TM can decide this language by:

   (a) Scanning left to right, marking each 0 and matching it with a 1
   (b) Accepting if all 0s and 1s are matched
   (c) Rejecting otherwise

   The TM always halts and gives a definite yes/no answer.

$$\begin{tikzcd}
0011 \in L \arrow[r] & \boxed{\text{TM for } L} \arrow[r] \arrow[dr] & \text{Accept} \\
001 \notin L \arrow[ur] & & \text{Reject}
\end{tikzcd}$$

Always halts with definite answer

2. Partially Decidable Languages

   A language L is partially decidable if there exists a TM that accepts all strings in L and either rejects or loops indefinitely on strings not in L. All partially decidable languages are recursively enumerable and vice versa.

   Example:

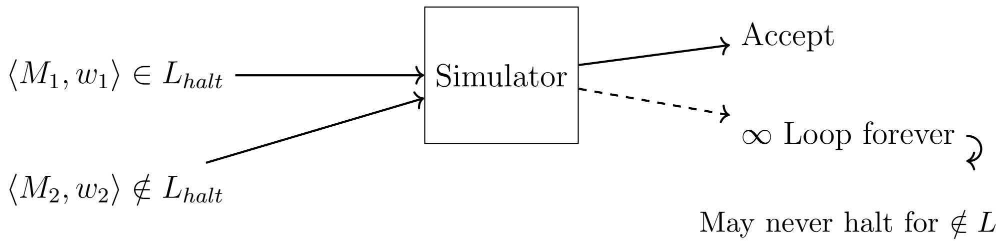
$$L_{halt} = \{\langle M, w\rangle \mid \text{TM } M \text{ halts on input } w\}$$

   This is the halting problem. A TM can simulate M on w:

   (a) If M halts on w, the simulator accepts (recognizes the string is in L_{halt})

   (b) If M loops forever on w, the simulator also loops forever (never rejects)

   We can recognize strings in the language but cannot decide the language.


$\langle M_1, w_1\rangle \in L_{halt}$
$\langle M_2, w_2\rangle \notin L_{halt}$
Simulator
Accept
∞ Loop forever
May never halt for $\notin L$

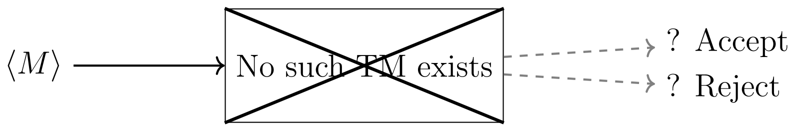
3. Undecidable Languages

   A language L is undecidable if there is no TM that can decide L. This means that no Turing machine can accept all strings in L and reject all strings not in L.

   Example:

$$L_{empty} = \{\langle M\rangle \mid \text{TM } M \text{ accepts no strings}\}$$

   This language asks whether a given TM's language is empty. By Rice's theorem, this is undecidable. There is no algorithm that can determine for arbitrary TM M whether $L(M) = \varnothing$. Any attempt to decide this would require solving the halting problem, which is impossible.



   Impossible to decide! No algorithm can solve this for all inputs

19.1 Halting Problem

The Halting Problem is the decision problem of determining whether a given program halts on a given input. Formally, let P be a program and I an input to that program. The halting problem asks whether $P(I)$ terminates in a finite number of steps.

Turing proved that there exists no algorithm H such that

$$H(P, I) = \begin{cases} 1, & \text{if program } P \text{ halts on input } I, \\ 0, & \text{if program } P \text{ does not halt on input } I. \end{cases}$$

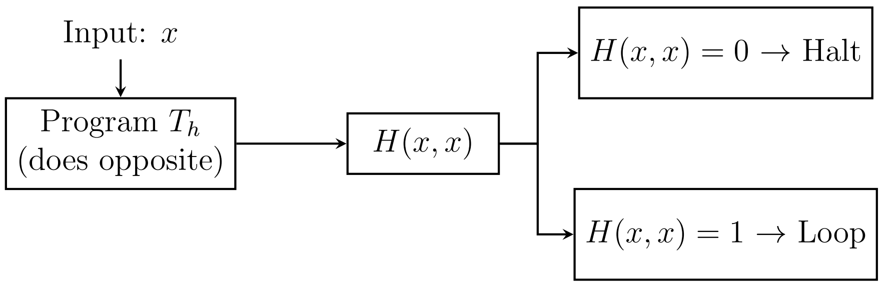
for all programs P and inputs I.

Example:

Define a program T_∞ such that

$$T_\infty(x) = \text{loop forever.}$$

Then $T_\infty(x)$ does not halt for any input x. Now define a program T_h as follows:



Running T_h on its own description leads to a contradiction:

1. If $H(T_h, T_h) = 1$, then by definition of T_h, it loops forever. This contradicts the assumption that H predicted it halts.

2. If $H(T_h, T_h) = 0$, then by definition of T_h, it halts. This contradicts the assumption that H predicted it does not halt.

Therefore, the Halting Problem is undecidable. No algorithm H can correctly decide whether all programs halt on all inputs.

19.2 Post Correspondence Problem (PCP)

The Post Correspondence Problem (PCP) is a decision problem defined as follows.

A PCP instance consists of a finite set of dominoes

$$P = \left\{ \frac{x_1}{y_1}, \frac{x_2}{y_2}, \ldots, \frac{x_n}{y_n} \right\},$$

where each $x_i, y_i \in \Sigma^*$ for some finite alphabet Σ. The problem asks whether there exists a nonempty sequence of indices

$$i_1, i_2, \ldots, i_k \quad (k \geq 1)$$

such that

$$x_{i_1}x_{i_2}\cdots x_{i_k} = y_{i_1}y_{i_2}\cdots y_{i_k}.$$

Example 1
Consider the following set of dominoes:

$$P = \left\{ \frac{1}{01}, \frac{10}{1}, \frac{0}{0} \right\}.$$

We want to determine whether this PCP instance has a solution. One possible sequence of indices is $[2, 1, 3]$:

$$\begin{aligned} x_2x_1x_3 &= \text{“10”} + \text{“1”} + \text{“0”} = \text{“1010”} \\ y_2y_1y_3 &= \text{“1”} + \text{“01”} + \text{“0”} = \text{“1010”} \end{aligned}$$

Since both concatenations are equal, this sequence is a solution, and the answer to this PCP instance is *yes*.
Example 2
Consider the following PCP instance:

$$P = \left\{ \frac{xy}{x}, \frac{yzx}{xy}, \frac{zx}{yzx} \right\}.$$

We test several sequences of indices.

1. Sequence $[1, 2, 3]$:

$$\begin{aligned} xy \cdot yzx \cdot zx &= \text{“xyyzxzx”} \\ x \cdot xy \cdot yzx &= \text{“xxyyzx”} \end{aligned}$$

   These strings are not equal.

2. Sequence $[2, 1, 3]$:

$$\begin{aligned} yzx \cdot xy \cdot zx &= \text{“yzxxyzx”} \\ xy \cdot x \cdot yzx &= \text{“xxyyzx”} \end{aligned}$$

   These strings are not equal.

3. Sequence $[3, 3, 1, 2]$:

$$\begin{aligned} zx \cdot zx \cdot xy \cdot yzx &= \text{“zxzxxyyzx”} \\ yzx \cdot yzx \cdot x \cdot xy &= \text{“yzxyzxxxy”} \end{aligned}$$

   These strings are not equal.

Hence, no solution has been found for this PCP instance.
Example 3
Consider the PCP instance

$$P = \left\{ \frac{x}{xx}, \frac{y}{yy} \right\}.$$

For any sequence of indices $[i_1, i_2, \ldots, i_k]$, suppose the sequence contains n occurrences of index 1 and m occurrences of index 2.

Then

$$\begin{aligned} x_{i_1}x_{i_2}\cdots x_{i_k} &= \text{a string of length } n+m, \\ y_{i_1}y_{i_2}\cdots y_{i_k} &= \text{a string of length } 2(n+m). \end{aligned}$$

Since the lengths of the two strings are different, no solution exists for this PCP instance.

Proof

We prove the undecidability of PCP by a reduction from the Halting Problem.Let $M = (W, \Sigma, \Gamma, V, q_0, q_{\text{accept}}, q_{\text{reject}})$ be a Turing machine and let $w \in \Sigma^*$ be an input string. The Halting Problem asks whether M halts on input w, which is known to be undecidable.

We construct a PCP instance

$$P = \left\{ \frac{u_1}{v_1}, \frac{u_2}{v_2}, \ldots, \frac{u_n}{v_n} \right\}$$

such that P has a solution if and only if M halts on input w.

A configuration of M is encoded as a string of the form $uqav$, where $u, v \in \Gamma^*$, $a \in \Gamma$, and $q \in W$. The dominoes of P are constructed to encode valid transitions between configurations of M.

1. An initial domino encoding the start configuration:

$$\frac{\#}{\#q_0w\#}.$$

2. For each symbol $a \in \Gamma \cup \{\#\}$, a copying domino:

$$\frac{a}{a}.$$

3. For each transition $V(q,a) = (p,b,R)$, a transition domino:

$$\frac{qa}{bp}.$$

4. For each transition $V(q,a) = (p,b,L)$ and for each $c \in \Gamma$, a transition domino:

$$\frac{cqa}{pcb}.$$

5. A final domino allowing acceptance:

$$\frac{q_{\text{accept}}}{q_{\text{accept}}}.$$

Any solution to the PCP instance corresponds to a valid sequence of configurations of M beginning with the initial configuration on input w and ending in an accepting configuration. Conversely, if M halts on input w, then the sequence of configurations of its computation yields a solution to the PCP instance. Therefore, the PCP instance P has a solution if and only if M halts on input w. Since the Halting Problem is undecidable, it follows that PCP is undecidable.