1. DEFINITION

1.1 DFA/REG

DFA

- **Definition**:

A **deterministic finite automaton** (DFA) is a 5-tuple

$$A = (Q, \Sigma, \delta, q_0, F)$$

where :

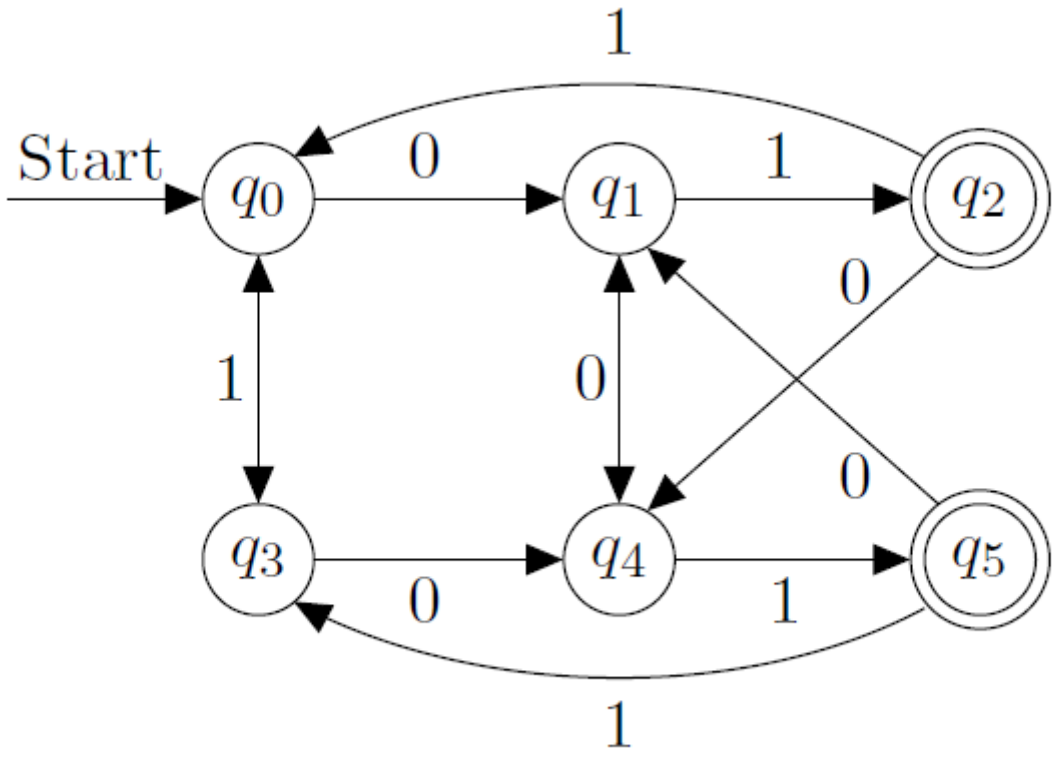
- Q is a **finite** set of **states**
- Σ is the input symbol **alphabet**
- δ is a **transition** function $Q \times \Sigma \rightarrow Q$
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is a set of final states

Equivalent Pairs

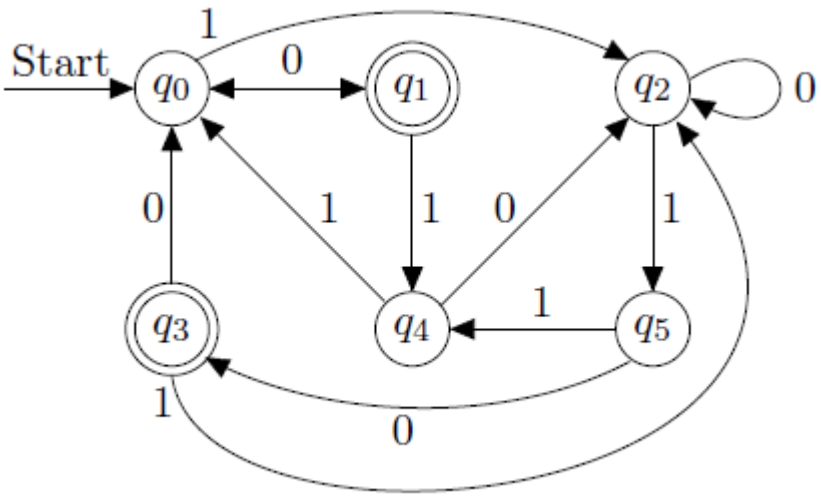
States p and q are equivalent if:

- For all input strings w, $\hat{\delta}(p, w)$ is an accepting state if and only if $\hat{\delta}(q, w)$ is an accepting state.

- **Example 1**: Intermediate steps to minimize



- **Idea**:
  - Each state has 0 and 1 direction.
  - Use intermediate steps/tabular algorithm/inductive algorithm/XYq table to find equivalent states.
  - Remove equivalent states to create a minimal DFA.
- **Solution**:
  - q0 = q3, q1 = q4 and q2 = q5.（等价符号）
- **Example 2**: Intermediate steps to minimize

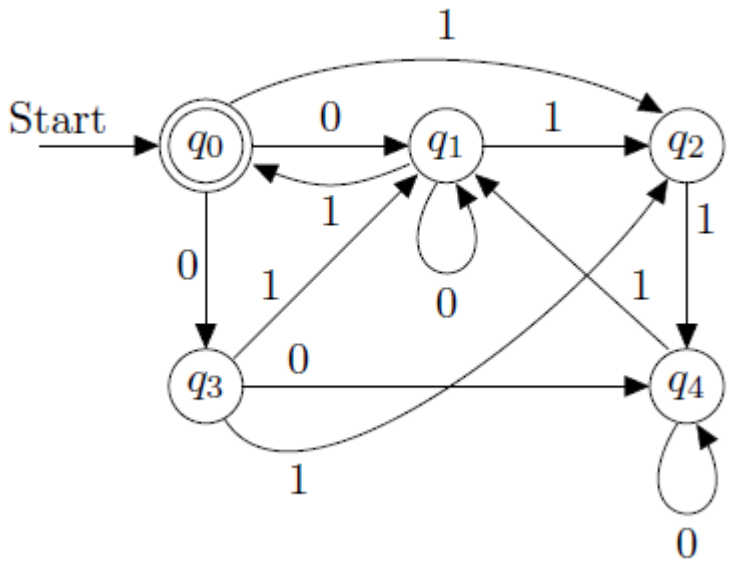


- **Solution**:

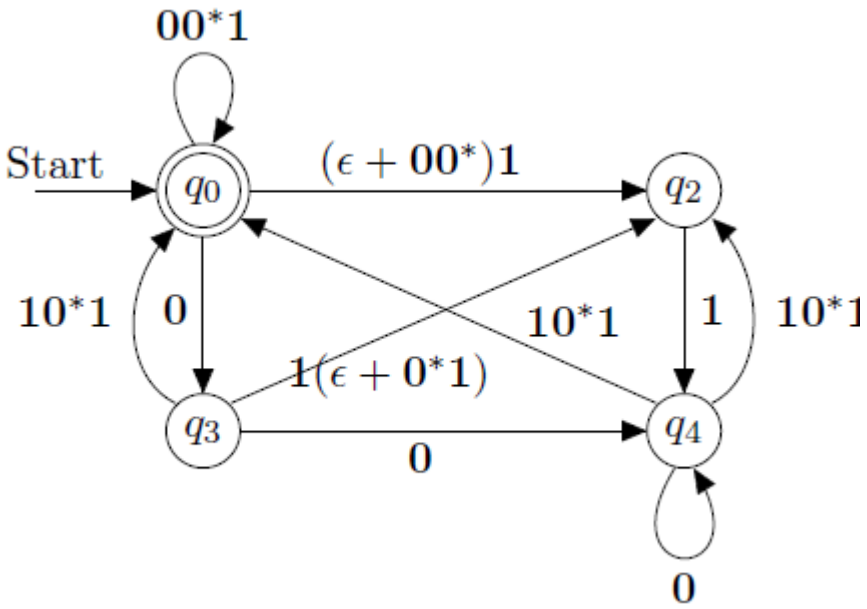
- p0 = {q0, q5}, p1 = {q1, q3}, p2 = {q2, q4}.

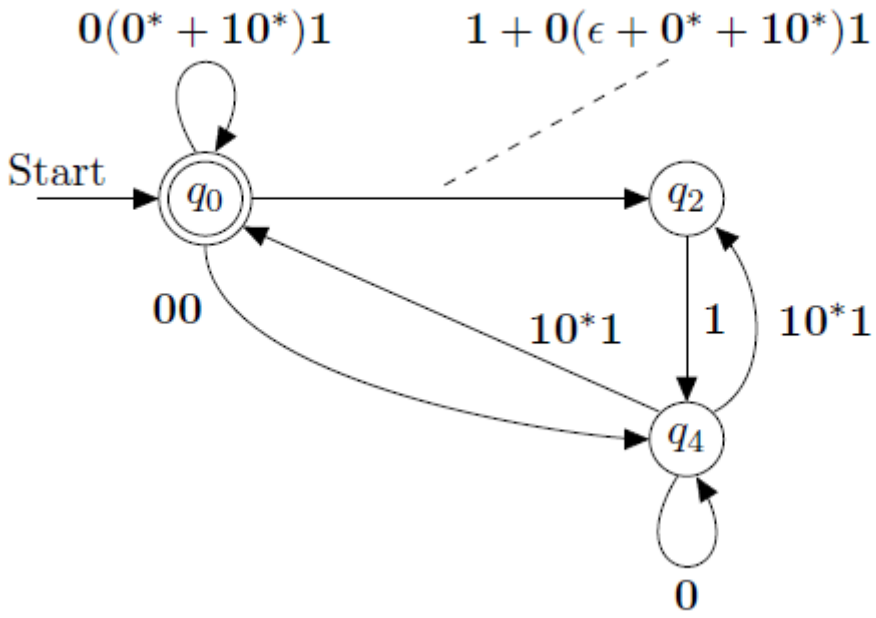
Regular Expression with ϵ-NFA

- **Examples 1**: Regular expression to minimize



- **Idea**:
  - Each state has 0 and 1 direction.
  - Eliminate the state using the regular expression.
- **Solution**:
  -



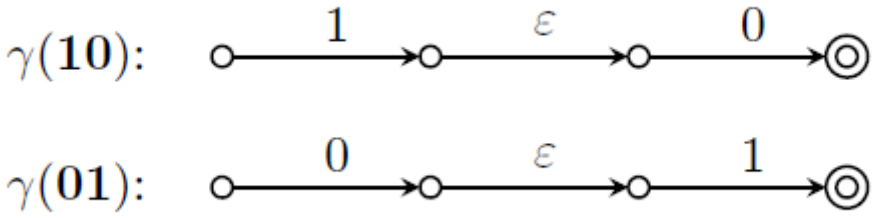


- **Examples 2**: Regular expression to ϵ-NFA

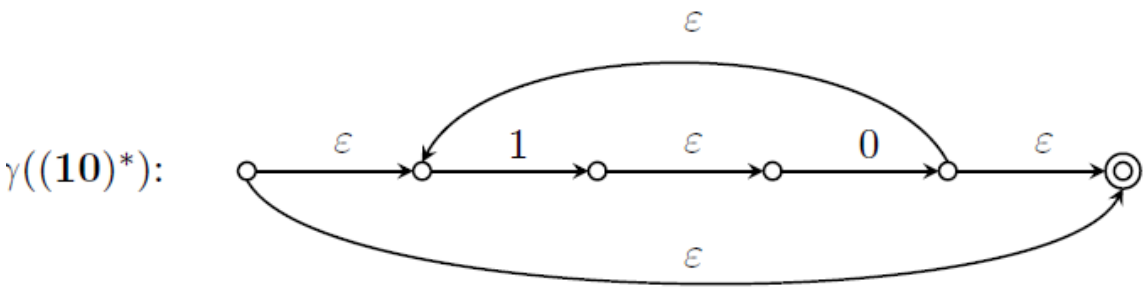
$$R = (10)^*\emptyset(\epsilon + 01)$$

- **Solution**:
  - Base:

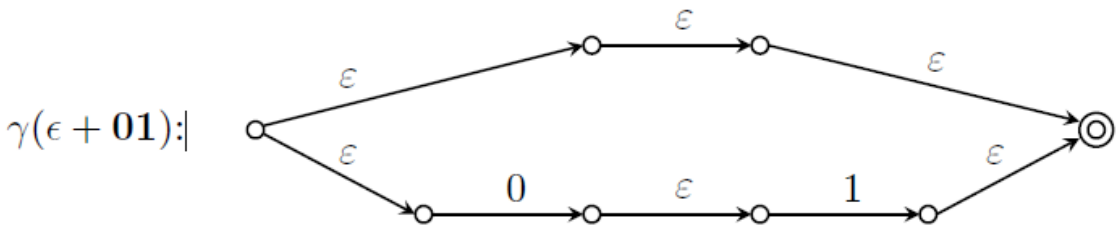
  - Length = 2 + ϵ



- (10)∗

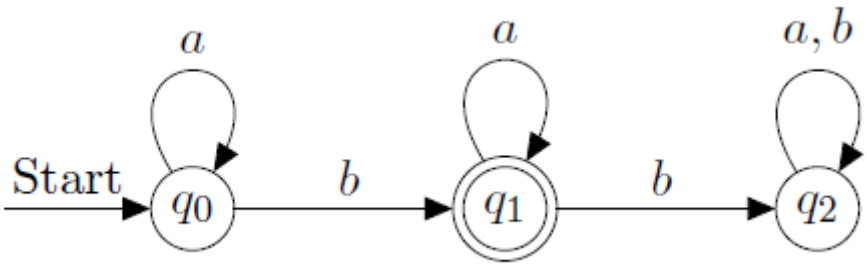


- γ(ϵ + 01):



- **Final Step**: R = (10)∗∅(ϵ + 01)

- **Induction**:
  - **Example**:



  - **Solution**:
    - Describe in words the language L(A).
    - Mutual induction on the length of x:
      - Base: |x| = 0, x = ε.
      - Induction: Let |x| = n > 0. Write x = yY, where Y ∈ {a, b}, y ∈ {a, b}∗, and |y| = n − 1. Distinguish two cases:
        - Case 1: Y = a.
        - Case 2: Y = b.

NFA

- **Definition**: NFA has many empty transitions.

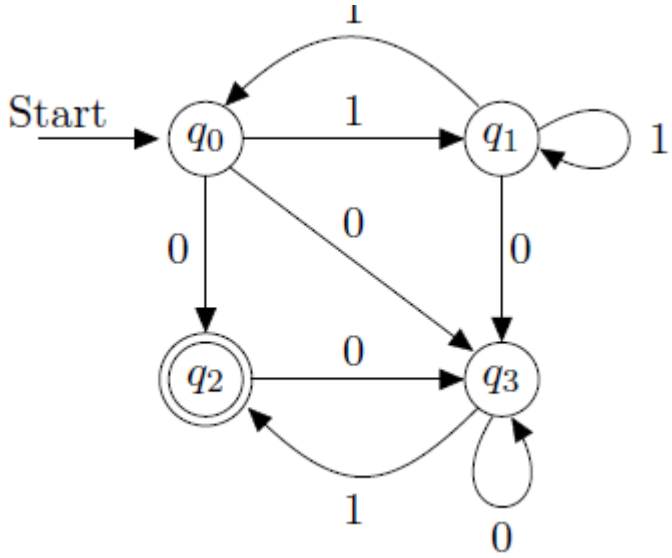
  A **nondeterministic finite automata** (NFA) is a 5-tuple

  $$A = (Q, \Sigma, \delta, q_0, F)$$

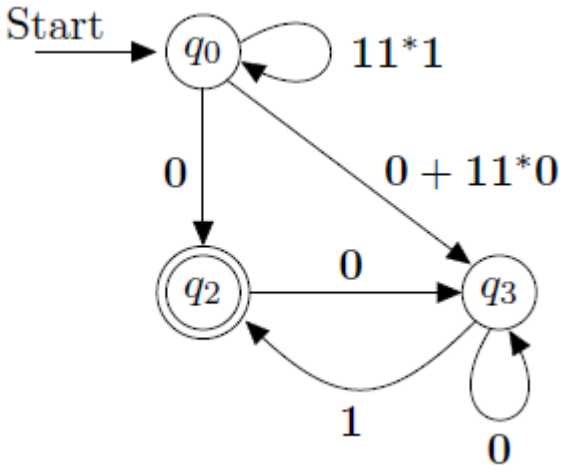
  where :

  - Q is a **finite** set of **states**
  - Σ is the **alphabet** of input symbols
  - δ is a **transition** function $Q \times \Sigma \rightarrow 2^Q$, where 2^Q is the set of all subsets of Q (power set)
  - $q_0 \in Q$ is the initial state
  - $F \subseteq Q$ is the set of final states

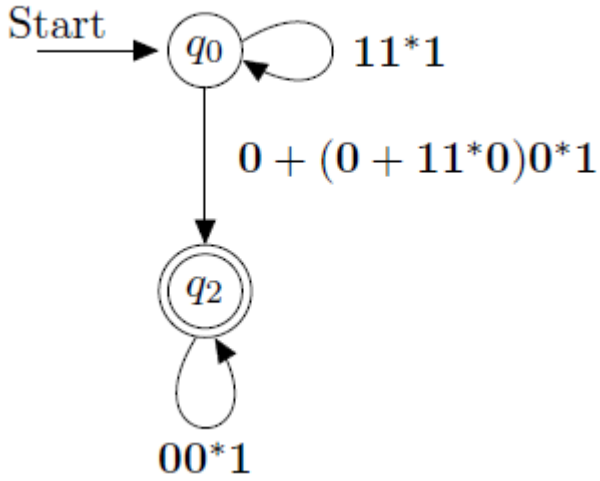
- **Example**:



- **Solution**: Use regular expression to eliminate state.
  - elimination of q1



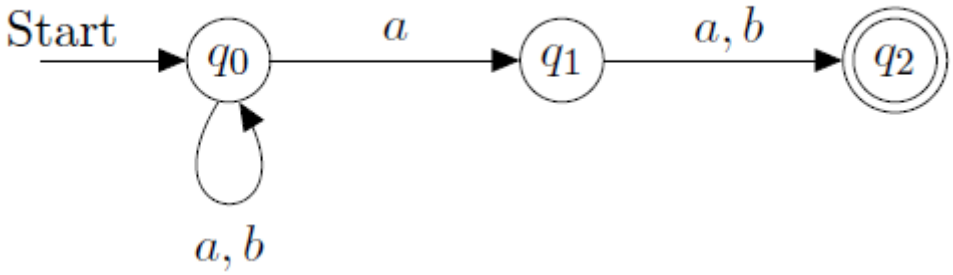
- elimination of q3



- final regular expression

$$\begin{aligned} E_{q_2} &= (11^*1 + (0 + (0 + 11^*0)0^*1)(00^*1)^*\emptyset)^*(0 + (0 + 11^*0)0^*1)(00^*1)^* \\ &= (11^*1 + \emptyset)^*(0 + (0 + 11^*0)0^*1)(00^*1)^* \\ &= (11^*1)^*(0 + (0 + 11^*0)0^*1)(00^*1)^* \ . \end{aligned}$$

- **Example 2**: Intermediate steps with induction



- **Assessment**:
  - Determine if the string "abab" belongs to the language L(N) by computing the value of ˆδN(q0, abab).

- Report all intermediate steps.

- **Solution**:

  This amounts to tracing each step of the computation of N on the input string $abab$
  - $\hat{\delta}_N(q_0, \epsilon) = \{q_0\}$
  - $\hat{\delta}_N(q_0, a) = \cup_{p\in\{q_0\}} \delta_N(p, a) = \delta_N(q_0, a) = \{q_0, q_1\}$
  - $\hat{\delta}_N(q_0, ab) = \cup_{p\in\{q_0,q_1\}} \delta_N(p, b) = \delta_N(q_0, b) \cup \delta_N(q_1, b) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$
  - $\hat{\delta}_N(q_0, aba) = \cup_{p\in\{q_0,q_2\}} \delta_N(p, a) = \delta_N(q_0, a) \cup \delta_N(q_2, a) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$
  - $\hat{\delta}_N(q_0, abab) = \cup_{p\in\{q_0,q_1\}} \delta_N(p, b) = \delta_N(q_0, b) \cup \delta_N(q_1, b) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

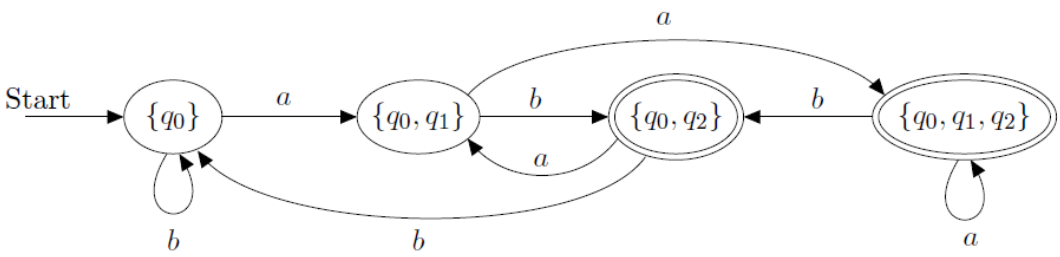
  Since q_2 is a final state for N, we conclude that the input string $abab$ is accepted by N.

- **Example 2**: Transform N into an equivalent deterministic finite automaton D. Depict the graphical representation of the function δD.

- **Solution**:

  - $\delta_D(\{q_0\}, a) = \{q_0, q_1\}$, $\delta_D(\{q_0\}, b) = \{q_0\}$
  - $\delta_D(\{q_0, q_1\}, a) = \{q_0, q_1, q_2\}$, $\delta_D(\{q_0, q_1\}, b) = \{q_0, q_2\}$
  - $\delta_D(\{q_0, q_2\}, a) = \{q_0, q_1\}$, $\delta_D(\{q_0, q_2\}, b) = \{q_0\}$
  - $\delta_D(\{q_0, q_1, q_2\}, a) = \{q_0, q_1, q_2\}$, $\delta_D(\{q_0, q_1, q_2\}, b) = \{q_0, q_2\}$

  The final states of D are $\{q_0, q_2\}$ and $\{q_0, q_1, q_2\}$. The graph representation of the function δ_D is reported below



1.2 CFL

- **PDA**:

Definition of push-down automaton

A **push-down automaton**, or PDA for short, is a tuple

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F),$$

with

- Q finite set of **states**
- Σ finite **input alphabet**
- Γ finite **stack alphabet**
- $\delta : Q \times \Sigma \cup \{\epsilon\} \times \Gamma \to 2^{Q \times \Gamma^*}$ is a **transition** function, always using **finite** subsets of $2^{Q \times \Gamma^*}$
- $q_0 \in Q$ is the initial state
- $Z_0 \in \Gamma$ is the initial stack symbol

  with no symbol in the stack δ is undefined
- $F \subseteq Q$ is the set of final states

- **Language Accepted by Final State**:
  - The **language accepted by final state** is the set of strings that take a pushdown automaton (PDA) from its initial configuration to a configuration where:
    - The input string is completely consumed.
    - The PDA is in a designated **final (accepting) state**.

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA

The **language accepted by final state** by P is

$$L(P) = \{w \mid (q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \alpha),\ q \in F\}$$

    - The stack does not necessarily need to be empty at the end of the computation.
    - The PDA cannot test the end of the string: this is an external condition in the definition of L(P).

- **Language Accepted by Empty Stack**:
  - The **language accepted by empty stack** is the set of strings that take a pushdown automaton (PDA) from its initial configuration to a configuration where:
    - The input string is completely consumed.

- The stack is **completely empty**.

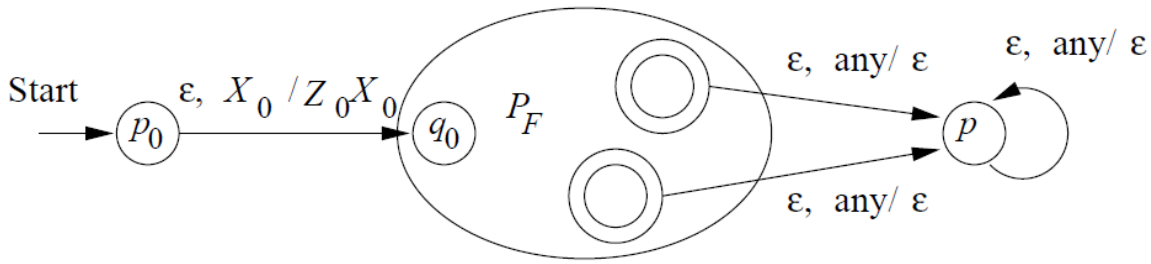
  Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be some PDA. The **language accepted by empty stack** by P is

  $$N(P) = \{w \mid (q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \epsilon)\}$$

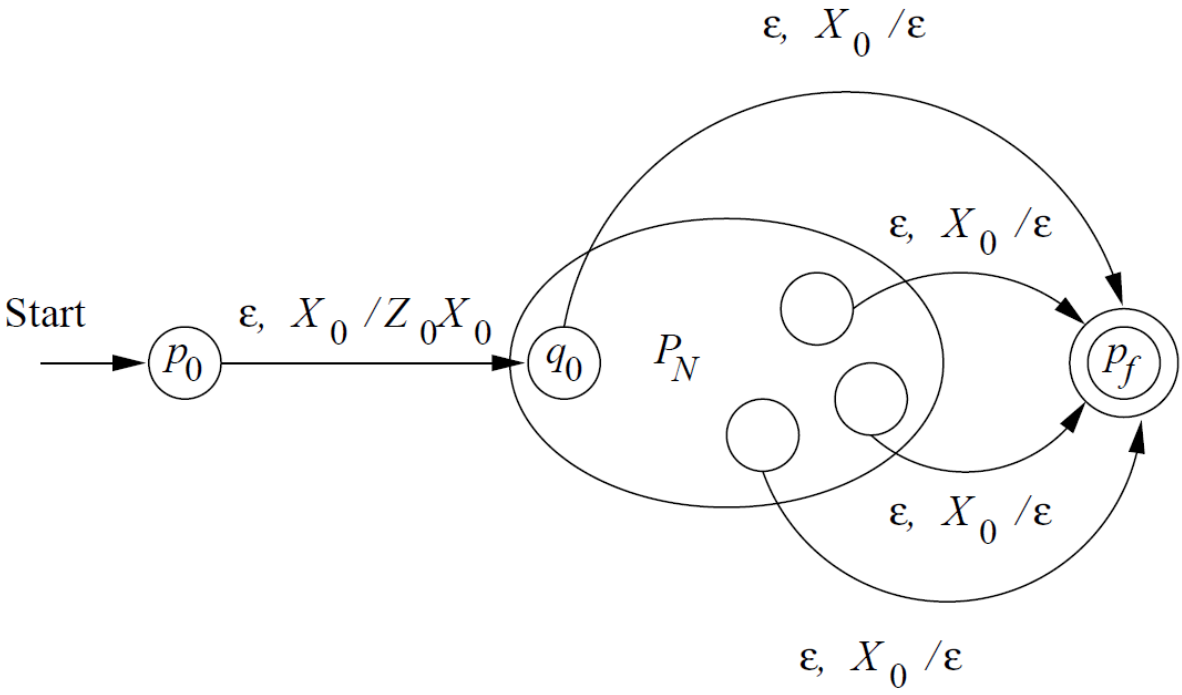
  for any state q

- Since final states are no longer relevant in this case, set F is not used in the definition.

- **From Empty Stack to Final State**:



- **From Final State to Empty Stack**:



- **Substitutions of CFL (7.3.1)**:

7.3.1 Substitutions

Let Σ be an alphabet, and suppose that for every symbol a in Σ, we choose a language L_a. These chosen languages can be over any alphabets, not necessarily Σ and not necessarily the same. This choice of languages defines a function s (a *substitution*) on Σ, and we shall refer to L_a as $s(a)$ for each symbol a.

Substitution

Assume two (finite) alphabets Σ and Δ, and a function

$$s : \Sigma \to 2^{\Delta^*}$$

Let $w \in \Sigma^*$, with $w = a_1 a_2 \cdots a_n$, $a_i \in \Sigma$. We define

$$s(w) = s(a_1).s(a_2).\cdots.s(a_n)$$

and, for $L \subseteq \Sigma^*$, we define

$$s(L) = \bigcup_{w \in L} s(w)$$

Function s is called a **substitution**

- related to the closure property of CFL. Some closure method is a form of substitution.

1.3 TM/RE/REC

- **TM**:

Turing machine

A **Turing machine**, MT for short, is a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

where

- Q is a finite set of **states**
- Σ is a finite set of **input symbols**
- Γ is a finite set of **tape symbols**, with $\Sigma \subseteq \Gamma$
- δ is a **transition function** from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$
- q_0 is the **initial state**
- $B \in \Gamma$ is the **blank symbol**, with $B \notin \Sigma$
- $F \subseteq Q$ is the set of **final states**

- **RE**: A language L is in RE if there exists a Turing machine M that halts and accepts for every string in L. If a string is not in L, the Turing machine may either reject or loop forever.

- **PCP**: The Post Correspondence Problem (PCP) is a well-known undecidable problem in computability theory. It involves two lists of strings and asks whether there is a way to arrange these strings such that the concatenations from the two lists match exactly.

An instance of **Post's correspondence problem**, or PCP for short, is formed by two **equal length** lists of strings

$$\begin{aligned} A &= w_1, w_2, \ldots, w_k \\ B &= x_1, x_2, \ldots, x_k \end{aligned}$$

where $w_i, x_j \in \Sigma^+$ and Σ is an alphabet with at least two symbols

Instance (A, B) has a solution if there are $m \geqslant 1$ indices $i_1, i_2, \ldots, i_m$ such that

$$w_{i_1} w_{i_2} \cdots w_{i_m} = x_{i_1} x_{i_2} \cdots x_{i_m}$$

- **Example**:

PCP instance with $\Sigma = \{0, 1\}$

	A	B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

A possible solution is provided by the indices:
$m = 4,\ i_1 = 2,\ i_2 = 1,\ i_3 = 1,\ i_4 = 3$

$$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3 = 101111110$$

Possible solutions are also all **repetitions** of 2,1,1,3

- **Theorem**: PCP is undecidable.

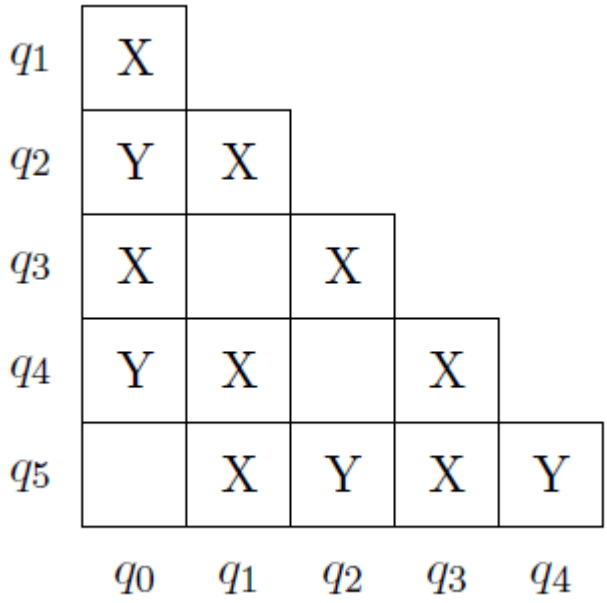
- **REC**:
  - A language L is recursive (REC) or, equivalently, the decision problem L represents is decidable, if L " LpMq for a TM M that halts for every input.
  - The class P (languages that can be recognized in polynomial time by a Turing machine) is a subset of the class REC (recursive languages).

---

2. GRAPH

2.1 DFA/REG

- **1.1 DFA**: XY-q Table - Inductive algorithm for detecting distinguishable state pairs



Question: Apply to A the tabular algorithm from the textbook for detecting pairs of equivalent states, reporting all the intermediate steps.

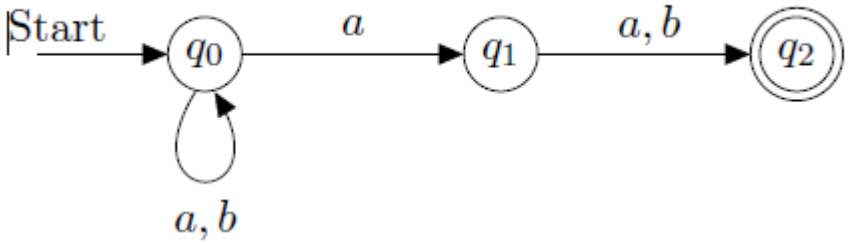
Thinking:

- Identify the minimal DFA and find the equal state pairs.
- P0, P1, P2 (final state). P0->P1 is Y, and the other 2 are X because they can go to the final state from the basis.

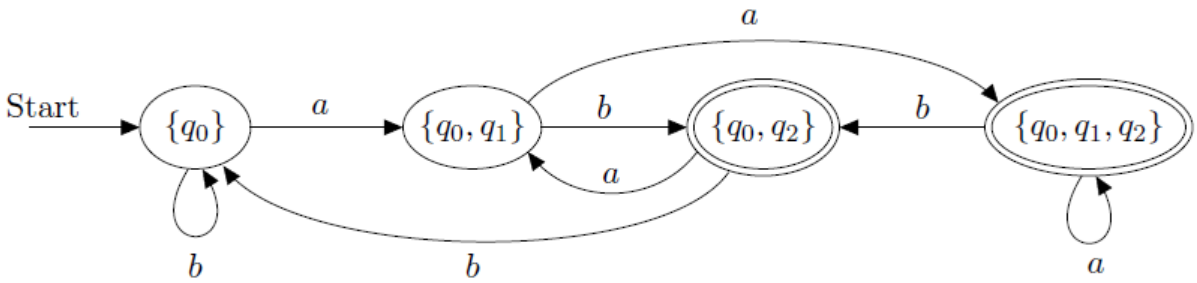
Solution:

- Marked with X the entries in the table corresponding to distinguishable state pairs detected in the base case of the algorithm, that is, state pairs that can be distinguished by the string 'epsilon'.
- Marked with Y distinguishable state pairs detected at the next iteration by some string of length one.
- At successive iterations, strings of length larger than one do not provide any new distinguishable state pairs.

- **1.2 NFA**: Lazy Evaluation



Lazy Evaluation:



---

2. CFL

- **2.1 CFL Production**:
  - **Example 1**:

$$L_1 = \{ba^m ba^n b \mid m, n \geq 1,\ m < n\}$$

Solution:

$$S \to bAb$$
$$A \to aAa \mid aBa$$
$$B \to Ba \mid ba$$

Explanation:

- 因为中间的那个b，的左右两边都有pumping，但是右边的a比左边的a多，所以构造最后的落脚点都在ba就可以保证右边的比左边多，这个就是B的逻辑。
- 然后A的两个pumping是因为=a本身的pumping（aAa）和B左右两边都要pumping（aBa）。
- 最后因为A包括了B，所以S是保证了首尾两个b的原因。

2.2 CFG Simplification

- **Elimination of**:
  - ϵ-productions
  - Unary productions
  - Useless symbols

Example 1

3. **[5 points]** Consider the CFG G implicitly defined by the following productions:

$$S \to ABA \mid BAB$$
$$A \to aA \mid bB$$
$$B \to b \mid \varepsilon$$

Perform on G the following transformations that have been specified in the textbook, in the given order. Report the CFGs obtained at each of the intermediate steps.

(a) Eliminate the ε-productions

(b) Eliminate the unary productions

(c) Eliminate the useless symbols

(d) Produce a CFG in Chomsky normal form equivalent to G.

Solution

- Eliminate the ϵ-productions

(a) The set of nullable variables of G is $n(G) = \{B\}$. After elimination of the ε-productions we obtain the intermediate CFG G_1

$$S \to ABA \mid AA \mid BAB \mid AB \mid BA \mid A$$
$$A \to aA \mid bB \mid b$$
$$B \to b$$

- Eliminate the unary productions

The only unary production in G_1 is $S \to A$. Thus the set of unary pairs of G_1 is

$$u(G_1) = \{(S, A)\} \cup \{(X, X) \mid X \in \{S, A, B\}\}.$$

After elimination of the unary productions we obtain the intermediate CFG G_2

$$S \to ABA \mid AA \mid BAB \mid AB \mid BA \mid aA \mid bB \mid b$$
$$A \to aA \mid bB \mid b$$
$$B \to b$$

- Eliminate the useless symbols

- Produce a CFG in Chomsky normal form equivalent to G.

  All nonterminals in G_2 are reachable and generating, that is, there are no useless nonterminals in G_2. Therefore this step does not change the intermediate CFG obtained at the previous step.

  The construction of a CFG in Chomsky normal form from G_2 proceeds in two steps. The first step eliminates terminal symbols in the right-hand side of the productions of G_2, in case they appear along with some other symbols. To do this we introduce new nonterminal symbols C_a, C_b and produce the intermediate CFG G_3

$$
\begin{aligned}
S &\to ABA \mid AA \mid BAB \mid AB \mid BA \mid C_aA \mid C_bB \mid b \\
A &\to C_aA \mid C_bB \mid b \\
B &\to b \\
C_a &\to a \\
C_b &\to b
\end{aligned}
$$

  The second step factorizes productions of G_3 having right-hand side of length larger than two. To do this we introduce new nonterminal symbols D, E and produce the final CFG G_4

$$
\begin{aligned}
S &\to AD \mid AA \mid BE \mid AB \mid BA \mid C_aA \mid C_bB \mid b \\
D &\to BA \\
E &\to AB \\
A &\to C_aA \mid C_bB \mid b \\
B &\to b \\
C_a &\to a \\
C_b &\to b
\end{aligned}
$$

2.3 Membership in a CFL

Given as input a string w, we want to decide whether w belongs to L(G), where G is some fixed CFG.

- **Dynamic Programming Algorithm**: Dynamic programming = table-filling algorithm = tabulation = CYK Algorithm.

  Prerequisite: Starts with a CNF grammar G =(V, T, P, S) for a language L.

1. Construct a triangular parse table and explain each element for the table, for example, for a 5-element string:

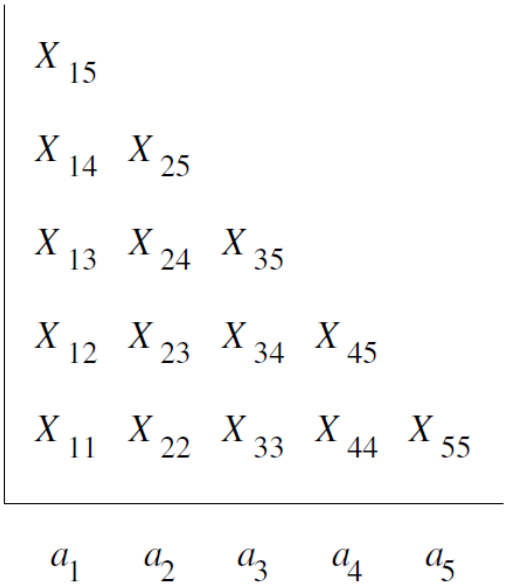


Figure 7.12: The table constructed by the CYK algorithm

Let $w = a_1 a_2 \cdots a_n$. We construct a triangular **parse table** where cell X_{ij} is set valued and contains all variables A such that

$$A \overset{*}{\underset{G}{\Rightarrow}} a_i a_{i+1} \cdots a_j$$

2. **Induction**:

Base $X_{ii} \leftarrow \{A \mid (A \rightarrow a_i) \in P\}$

Induction We build X_{ij} for increasing values of $j - i \geqslant 1$

$X_{ij} \leftarrow X_{ij} \cup \{A\}$ if and only if there exist k, B, C such that

- $i \leqslant k < j$
- $(A \rightarrow BC) \in P$
- $B \in X_{ik}$ and $C \in X_{k+1,j}$

Example 1:

abcde				
abcd	bcde			
abc	bcd	cde		
ab	bc	cd	de	
a	b	c	d	e

然后根据PDA，推出怎么样得出这样的string。

Example 2:

3. **[6 points]** With reference to the membership problem for context-free languages, answer the following two questions.

   (a) Specify the dynamic programming algorithm reported in the textbook for the solution of this problem.

   (b) Consider the CFG G in Chomsky normal form defined by the following rules:

$$
\begin{aligned}
S &\to CD \\
C &\to AC' \mid c \\
C' &\to CB \\
A &\to a \\
B &\to b \\
D &\to DD \mid d
\end{aligned}
$$

   Assuming as input the CFG G and the string $w = aacbbdddd$, trace the application of the algorithm in (a).

- **dynamic programming algorithm**

$\{S\}$								
$\{S\}$								
$\{S\}$								
$\{S\}$								
$\{C\}$								
	$\{C'\}$				$\{D\}$			
	$\{C\}$				$\{D\}$	$\{D\}$		
		$\{C'\}$			$\{D\}$	$\{D\}$	$\{D\}$	
$\{A\}$	$\{A\}$	$\{C\}$	$\{B\}$	$\{B\}$	$\{D\}$	$\{D\}$	$\{D\}$	$\{D\}$
a	a	c	b	b	d	d	d	d

Example

(b) Consider the CFG G defined by the following rules:

$$
\begin{aligned}
S &\to BC \\
B &\to BB \mid b \\
C &\to BC \mid c
\end{aligned}
$$

Assuming as input the CFG G and the string $w = bbbbc$, trace the application of the above algorithm.

Solution

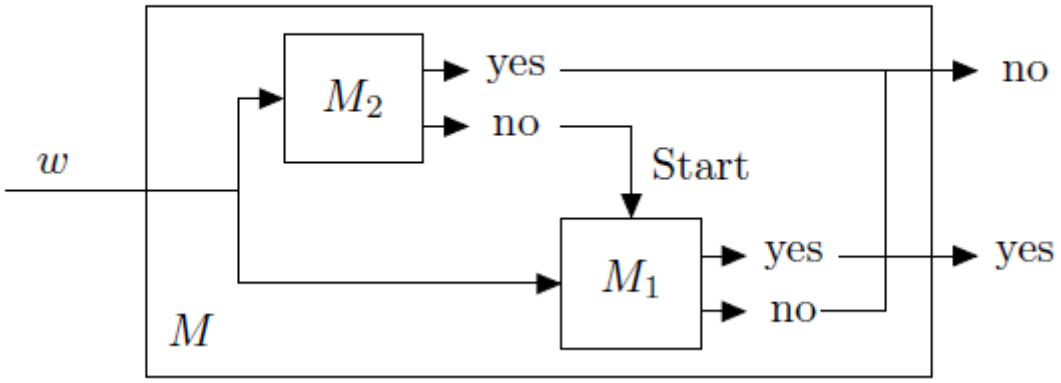
$\{S, C\}$				
$\{B\}$	$\{S, C\}$			
$\{B\}$	$\{B\}$	$\{S, C\}$		
$\{B\}$	$\{B\}$	$\{B\}$	$\{S, C\}$	
$\{B\}$	$\{B\}$	$\{B\}$	$\{B\}$	$\{C\}$
b	b	b	b	c

---

3. TM/RE/REC

3.1 REC

Given two languages L1 and L2 in REC, the language L1 r L2 is always in REC. Difference：



判定题

---

1. Closure Property

Operation	CFL	REG	RE	REC
Union	Yes	Yes	Yes	Yes

Operation	CFL	REG	RE	REC
∩ Intersection	**NO**	Yes	Yes	Yes
Complementation	**NO**	Yes	NO	Yes
Concatenation	Yes	Yes	Yes	Yes
Kleene Star	Yes	Yes	Yes	
Difference	**NO**	Yes	NO	
Homomorphism同态	Yes	Yes		
Reversal	Yes	Yes		
空	Yes	Yes		

Operation OF CFL and RL	RESULT
CFL ∩ REG	CFL
CFL ＼ REG	CFL

Operation	REC	Usage
Complement	Yes	If L is in RE and L-(Complement) is not in RE, then L cannot be a recursive language.
		If L and L-(Complement) are in RE, then L is recursive.
		It is not possible that a language is recursive and the complement is RE but not recursive or not RE.
		It is not possible that a language and its complement are both RE but not recursive.

Proof:

CFL: 倒插门，戴帽子，来一刀 都不行

- 倒插门有很简单的反例
- 因为倒插门，所以证明戴帽子不行
- 因为戴帽子，所以来一刀不行

- **Intersection**: CFL 不闭合

CFL & intersection

$L_1 = \{0^n 1^n 2^i \mid n \geqslant 1,\ i \geqslant 1\}$ is a CFL, generated by the CFG

$$S \to AB$$
$$A \to 0A1 \mid 01$$
$$B \to 2B \mid 2$$

$L_2 = \{0^i 1^n 2^n \mid n \geqslant 1,\ i \geqslant 1\}$ is a CFL, generated by the CFG

$$S \to AB$$
$$A \to 0A \mid 0$$
$$B \to 1B2 \mid 12$$

$L_1 \cap L_2 = \{0^n 1^n 2^n \mid n \geqslant 1\}$ which is not a CFL

- **CFL Complementation L- no**:

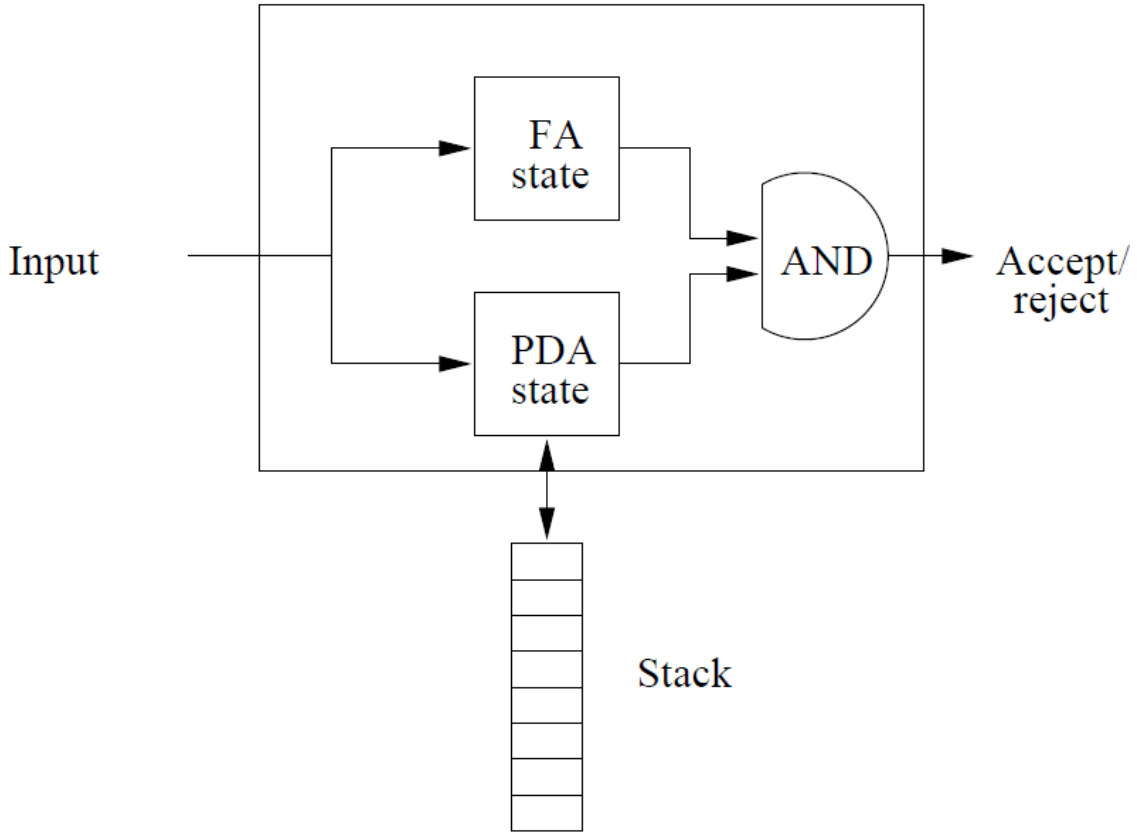
$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

= create contradiction of intersection + Intersection NO closure

- **CFL1** $\setminus$ **CFL2**: = create contradiction of complementation using special case

$$\Sigma^* \setminus L = \overline{L}$$

- **Intersection between CFL and Regular Language**:



（这个就是CFL倒插门REG的特殊情况，但是还是CFL）

理解:

- CFL ∩ CFL 可能需要多个计数器（比如 a^n b^n c^n 需要两个），超出了 CFL 的能力范围，因此不一定是 CFL。
- CFL ∩ 正则语言：正则语言只提供有限的约束，不会让 CFL 变得更复杂，因此交集仍然是 CFL。
- **CFL ⧹ REG**:

Proof

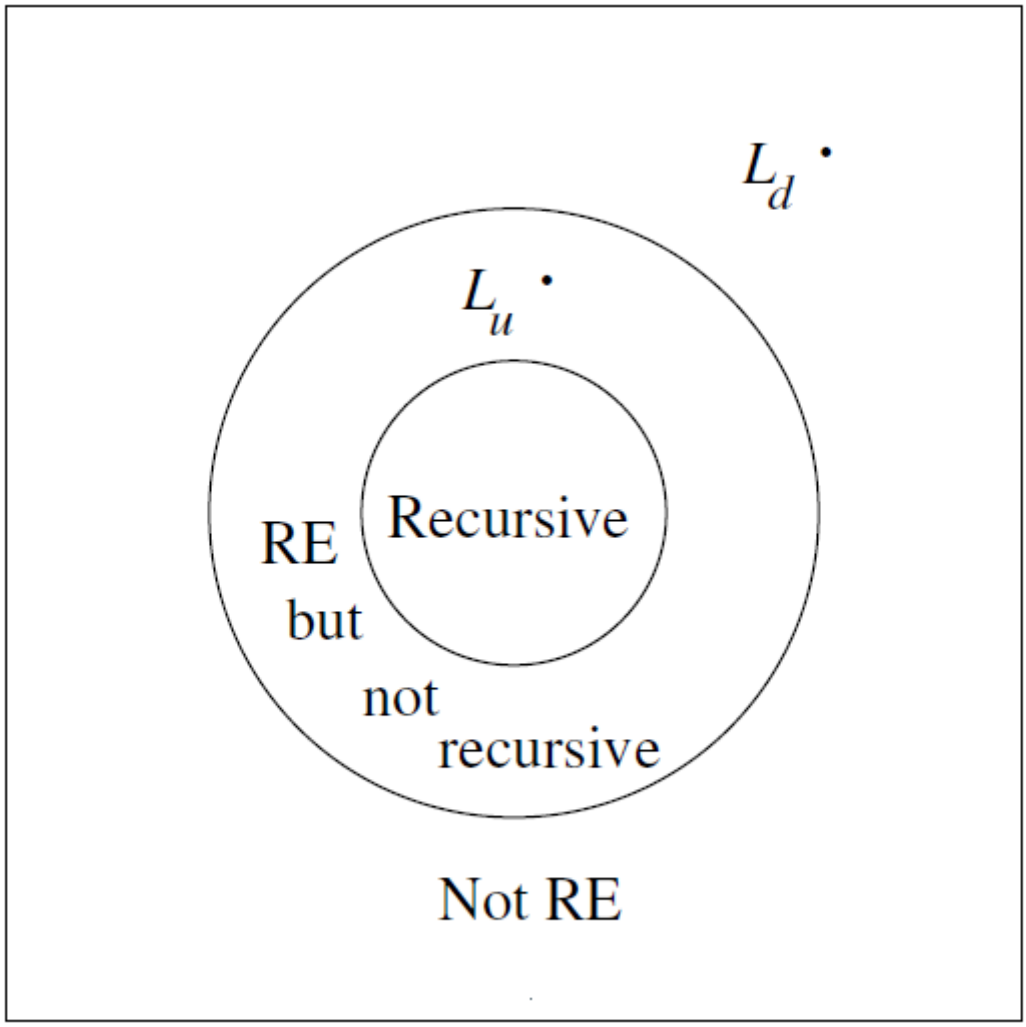
Operator $\setminus$ *with REG* : $\overline{R}$ is regular, $L \cap \overline{R}$ is CFL, and $L \cap \overline{R} = L \setminus R$

= REG is CFL + REG complementation is CFL + intersection of 2 CFL （这个就是CFL⧹ CFL的特殊情况，但是还是CFL）

Example:

- We have used Rice's theorem to prove Lp is not REC.
- Then, how to find if L is RE or not?
  - **NOT EMPTY**: We use the Lp- to prove it is RE. (We can construct a TM for this property)
  - **NOT TRIVIAL**: If Lp is also RE, it Lp should be REC. But we have proved it's not REC. So Lp can't be RE. It proves that Lp is also not RE.

2. RE REC TM



- **Lu** is in RE but is not recursive. (use contradiction and reduction)

  The language L_u, called **universal language**, is the set

  $$L_u = \{\text{enc}(M, w) \mid w \in L(M)\}.$$

  In words, L_u is the set of binary strings that encode a pair (M, w) such that $w \in L(M)$.

- **Ld** is not in RE. (use contradiction to prove this)

  The **diagonalization language** is the set

  $$L_d = \{w \mid w = w_i,\ w_i \notin L(M_i)\}$$

- **Lh** is a RE language, Lh is not recursive.

Let us consider the language L_h, called the **halting problem**

$$L_h = \{\text{enc}(M, w) \mid w \in H(M)\} .$$

- **Lne** is RE. Lne is non-recursive.
- **Le** is not in RE. (use theorem: If L and L- are in RE, then L is recursive.)

We consider two languages formed by TM encodings

$$L_e = \{\text{enc}(M) \mid L(M) = \varnothing\} ;$$
$$L_{ne} = \{\text{enc}(M) \mid L(M) \neq \varnothing\} .$$

Note : $\overline{L_e} = L_{ne}$.

Pumping Lemma

- A language is not a regular language.
- A language is not CFL.
- Regular language can use regular grammar or can prove the language is a finite language.
- CFL can use PDA to represent or can use the grammar to present.

REC/RE

Rice's Theorem for Undecidability

A property is non-trivial to prove not REC.

- **NOT EMPTY**: A case suits the language property.
- **NOT RE/P not RE/RE not P**: A case that is not decidable.

Examples:

- **Example 1**:

5. **[7 points]** Let R be the string reversal operator, extended to languages as usual. Consider the following property of the RE languages defined over the alphabet $\Sigma = \{0, 1\}$

$$\mathcal{P} \quad = \quad \{L \mid L \in \text{RE},\ L \cap L^R = \emptyset\}$$

where the condition $L \cap L^R = \emptyset$ means that for every string $w \in L$, w^R does not belong to L. Define $L_{\mathcal{P}} = \{\text{enc}(M) \mid L(M) \in \mathcal{P}\}$.

(a) Use Rice's theorem to show that $L_{\mathcal{P}}$ is not in REC.

(b) State whether $L_{\mathcal{P}}$ is in RE$\setminus$REC or else outside of RE.

Solution

(a) We have to show that property $\mathcal{P}$ is not trivial.

- $\mathcal{P} \neq \emptyset$. Consider the language $L = \{1100\}$. Since L is finite, L is also in RE. Observe that $L^R = \{0011\}$ and $L \cap L^R = \emptyset$. Therefore $L \in \mathcal{P}$.
- $\mathcal{P} \neq \text{RE}$. Consider the language $L = \{1100, 0011\}$. Since L is finite, L is also in RE. Observe that $L \cap L^R = L \neq \emptyset$, and therefore $L \notin \mathcal{P}$.

- **Example 2**:

Let M be a TM defined over the input alphabet $\Sigma = \{0, 1\}$, and let enc(M) be some binar encoding of M. Consider the languages

$$L_1 \quad = \quad \{\text{enc}(M) \mid \text{there is some input such that } M \text{ accepts in exactly 5 steps}\}\ ,$$
$$L_2 \quad = \quad \{\text{enc}(M) \mid \text{there is some input of length 5 that is accepted by } M\}\ .$$

Proof:

Language L_2 does not belong to REC. To see this, we define a property of the RE languages $\mathcal{P} = \{L \mid L \in \text{RE}$, there exists some $w \in L$ such that $|w| = 5\}$. We now define $L_{\mathcal{P}} = \{\text{enc}(M) \mid L(M) \in \mathcal{P}\}$, and observe that $L_{\mathcal{P}} = L_2$. We can then apply Rice's theorem and show that $\mathcal{P}$ is not trivial. First, Σ^* is in RE and contains some string w such that $|w| = 5$. Therefore we have $\Sigma^* \in \mathcal{P}$ and $\mathcal{P}$ is not empty. Second, the empty language $\emptyset$ is in RE and does not contain any string w such that $|w| = 5$. Therefore we have $\emptyset \notin \mathcal{P}$, and thus $\mathcal{P}$ does not contain every RE language. Since $\mathcal{P}$ is not trivial, we can conclude that L_2 is not in REC, according to Rice's theorem.

- **Example 3**:

Let $L_1 = \{0^n 1^n \mid n \geq 1\}$. Define the following property of the RE languages

$$\mathcal{P} \quad = \quad \{L \mid L \in \text{RE},\ L \cap L_1 = \emptyset\}\ ,$$

Proof:

Solution The property $\mathcal{P}$ is not trivial. There are several ways to prove this. For instance, the language $L = \emptyset$ is a regular language and therefore also a RE language. We have $L \in \mathcal{P}$, since $L \cap L_1 = \emptyset$. Furthermore, the language $L' = \{0, 1\}^*$ is a regular language and therefore also a RE language. We have $L' \notin \mathcal{P}$, since $L' \cap L_1 = L_1 \neq \emptyset$. By applying Rice's theorem, we can conclude that $L_{\mathcal{P}}$ is not in REC.

- **Example 4**:

[**9 points**] For a property $\mathcal{P}$ of the RE languages, define $L_{\mathcal{P}} = \{\mathsf{enc}(M) \mid L(M) \in \mathcal{P}\}$.

(a) Let k be some fixed natural number with $k > 1$. Consider the following properties of the RE languages defined over the alphabet $\Sigma = \{0, 1\}$:

$$\begin{aligned} \mathcal{P}_{<k} &= \{L \mid L \in \text{RE}, |L| < k\}\,; \\ \mathcal{P}_{\geq k} &= \{L \mid L \in \text{RE}, |L| \geq k\}\,. \end{aligned}$$

Proof:

Language $L_{\mathcal{P}_{\geq k}}$ is not in REC. To prove this statement, we apply Rice's theorem and show that property $\mathcal{P}_{\geq k}$ is not trivial. First, Σ^* is in RE and has more than k strings. Therefore we have $\Sigma^* \in \mathcal{P}_{\geq k}$ and $\mathcal{P}_{\geq k}$ is not empty. Second, the empty language $\emptyset$ is in RE and has fewer than k strings, since $k \geq 1$. Therefore we have $\emptyset \notin \mathcal{P}_{\geq k}$, and $\mathcal{P}_{\geq k}$ does not contain every RE language. Since $\mathcal{P}_{\geq k}$ is not trivial, we can conclude that $L_{\mathcal{P}_{\geq k}}$ is not in REC, according to Rice's theorem. We now prove that $L_{\mathcal{P}_{\geq k}}$ is in RE. To this end, we specify a nondeterministic TM N such that $L(N) = L_{\mathcal{P}_{\geq k}}$. Let $\mathsf{enc}(M)$ be the input to N.

- **Example 5**:

5. [**9 points**] Consider the following property of the RE languages defined over the alphabet $\Sigma = \{0, 1\}$:

$$\mathcal{P} = \{L \mid L \in \text{RE}, \text{ every string in } L \text{ has prefix } 111\}$$

and define $L_{\mathcal{P}} = \{\mathsf{enc}(M) \mid L(M) \in \mathcal{P}\}$.

Proof:

We apply Rice's theorem and show that $\mathcal{P}$ is not trivial. First, the language $\emptyset$ is in RE and we have already shown that $\emptyset \in \mathcal{P}$. Therefore we have $\mathcal{P} \neq \emptyset$. Second, the language $\{101\}$ is in RE and does not belong to $\mathcal{P}$. Therefore we have $\mathcal{P} \neq \text{RE}$. We then conclude that $L_{\mathcal{P}}$ is not in REC.

- **Example 6**:

[**7 points**] Consider the following property of the RE languages defined over the alphabet $\Sigma = \{0, 1\}$

$$\mathcal{P} = \{L \mid L \in \text{RE}, \text{ for every pair } u, v \in L \text{ we have } u \cdot v \notin L\}$$

Proof:

Solution Language $L_{\mathcal{P}}$ is not in REC. To prove this, we use Rice's theorem and show that property $\mathcal{P}$ is not trivial. First, consider the finite language $L_1 = \{01, 10, 11, 00\}$, which is in RE. Since every string in L_1 has length 2, it follows that the concatenation of every pair of strings from L_1 provides a string of length 4, which is not in L_1. Therefor L_1 has the property $\mathcal{P}$, and thus $\mathcal{P}$ is not empty. Second, consider the finite language $L_2 = \{0, 00, 000\}$, which is in RE. For the pair of strings $0, 00 \in L_2$ we have $0 \cdot 00 = 000 \in L_2$. Therefor L_2 does not have the property $\mathcal{P}$, and thus $\mathcal{P}$ is not the whole class RE. We then conclude that property $\mathcal{P}$ is not trivial.

- **Example 7**:

[**7 points**] Let R be the string reversal operator, extended to languages as usual. Consider the following property of the RE languages defined over the alphabet $\Sigma = \{0, 1\}$

$$\mathcal{P} = \{L \mid L \in \text{RE}, L \cap L^R = \emptyset\}$$

Proof:

We have to show that property $\mathcal{P}$ is not trivial.

- $\mathcal{P} \neq \emptyset$. Consider the language $L = \{1100\}$. Since L is finite, L is also in RE. Observe that $L^R = \{0011\}$ and $L \cap L^R = \emptyset$. Therefore $L \in \mathcal{P}$.
- $\mathcal{P} \neq \text{RE}$. Consider the language $L = \{1100, 0011\}$. Since L is finite, L is also in RE. Observe that $L \cap L^R = L \neq \emptyset$, and therefore $L \notin \mathcal{P}$.

- **Example 8**:

  Language L_1 is not in REC. To show this, consider the following property of the RE languages

  $$\mathcal{P} \quad = \quad \{L \mid L \in \text{RE}, \ |L| = 5\}$$

  Language L_1 is not in REC. To show this, consider the following property of the RE languages

  $$\mathcal{P} \quad = \quad \{L \mid L \in \text{RE}, \ |L| = 5\}$$

  and observe that $L_1 = L_{\mathcal{P}} = \{\text{enc}(M) \mid L(M) \in \mathcal{P}\}$. We can now apply Rice's theorem and show that property $\mathcal{P}$ is not trivial.

  - $\mathcal{P} \neq \emptyset$. Consider the language $L = \{\epsilon, 0, 1, 01, 10\}$ which is in RE, and observe that $L \in \mathcal{P}$.
  - $\mathcal{P} \neq \text{RE}$. Consider the language $\{0, 1\}^*$ which is in RE, and observe that $L \notin \mathcal{P}$.

Reduction

Format:

- To specify a reduction LP ≤m L, we need to describe a mapping from strings enc(M) [Lp] to strings enc(M1, M2) [L] with the property that **enc(M) ∈ LP** if and only if the associated string **enc(M1, M2) ∈ L**.
- Our mapping then sets M1 = M... and M2 = ...

Examples:

- **Example 1**:

  Let M_1 and M_2 be TMs defined over the input alphabet $\Sigma = \{0, 1\}$, and let $\text{enc}(M_1, M_2)$ be some binary encoding of M_1 and M_2. Consider the language

  $$L_3 \quad = \quad \{\text{enc}(M_1, M_2) \mid \overline{L(M_1)} = L(M_2)\} \ ,$$

  where $\overline{L}$ is the complement of language L with respect to Σ^*. Assess whether L_3 belongs to the classes RE or not.

  Solution:

  - This one is not related to the previous question, so I think of something else.
  - It reminds me of the le and lne. In the lecture, Lne is RE but not REC (reduction from Lu), Le is not in RE (if they are, they should be REC, but lne is not REC).
  - So I use the reduction of the Le<m L3.
  - We need to map enc(M) ∈ Le,
    - iff definition of Le (first thing first, let the dog out): L(M)=空（这里的M就是Le，用Le定义这个M）

- definition of complementation (the property we use): L(M)的反（因为下面的reduction要用）
- definition of reduction (Le和L3的关系): 从题目中摘取这个reduction（上面已经用了M，下面就用M1，M2，刚好和题目里面一样）
- definition of L3 (last thing is the L we want to get): enc(M1, M2) ∈ L3（最后得到我们的enc(M)属于L3，就是Le<m L3）

L_3 is not in RE. To show this, we consider the language L_e, that is, the language of the encodings of all TMs that accept the empty language, and show a reduction $L_e \leq_m L_3$. Since L_e is not in RE, the reduction proves the desired claim.

We need to map instances $\mathsf{enc}(M)$ of L_e into instances $\mathsf{enc}(M_1, M_2)$ of L_3. Let M_{Σ^*} be a TM such that $L(M_{\Sigma^*}) = \Sigma^*$. We set $M_1 = M$ and $M_2 = M_{\Sigma^*}$. The following chain of logical equivalences shows that the construction represents a valid reduction:

$$\begin{array}{lll} \mathsf{enc}(M) \in L_e & \text{iff} \quad L(M) = \emptyset & \text{(definition of } L_e\text{)} \\ & \text{iff} \quad \overline{L(M)} = \Sigma^* & \text{(definition of complementation)} \\ & \text{iff} \quad \overline{L(M_1)} = L(M_2) & \text{(definition of our reduction)} \\ & \text{iff} \quad \mathsf{enc}(M_1, M_2) \in L_3 & \text{(definition of } L_3\text{)} \, . \end{array}$$

- **Example 2**:

[7 points] Define the following property of the RE languages defined over the alphabet $\Sigma = \{0, 1\}$

$$\mathcal{P} \quad = \quad \{L \mid L \in \text{RE}, \; L \neq \Sigma^*\} \, .$$

Assess whether the language $L_{\mathcal{P}}$ belongs to the class REC, and provide a mathematical proof of your answer. Consider also the following language

$$L \quad = \quad \{\mathsf{enc}(M, M') \mid L(M) \cap L(M') \neq \Sigma^*\}$$

where M, M' are generic TMs accepting languages defined over Σ, and $\mathsf{enc}(M, M')$ is a binary string representing a fixed encoding for M, M'. Assess whether the language L belongs to the class REC, and provide a mathematical proof of your answer.

Solution:

- The first part has used Rice's theorem to prove that Lp is not REC. The second question needs to use reduction to prove L is REC or not.
- Since a string in L is not the encoding of a single TM, it is instead the encoding of a pair of TMs. Two M in the definition.
- We find that the structure of L is the same as Lp (≠), we will use it in definition to construct the reduction.
- **Reduction**: Lp ∈ L, enc(M) ∈ Lp
- Iff
  - Lp definition: L(M) ∈ Lp
  - property: L(M) n L(M) ≠元素的集合， 这个是我们要用到的一个性质
  - reduction of Lp and L: Lp ∈ RE, here M已经表示了Lp，我们用M' M''表示我们的L，L(M') \ L(M'') ≠元素的集合
  - L definition: enc(M) ∈ L: the reduction format enc(M', M'') ∈ L

$\mathsf{enc}(M', M'')$ such that $\mathsf{enc}(M) \in L_{\mathcal{P}}$ if and only if $\mathsf{enc}(M', M'') \in L$. To do this, we simply set $M' = M$ and $M'' = M$. In this way, we have the following chain of logical equivalences, which proves that the construction is a valid reduction

$$\begin{array}{lll} \mathsf{enc}(M) \in L_{\mathcal{P}} & \text{iff} \quad L(M) \neq \Sigma^* & \text{(definition of } L_{\mathcal{P}}\text{)} \\ & \text{iff} \quad L(M) \cap L(M) \neq \Sigma^* & \text{(definition of } \cap\text{)} \\ & \text{iff} \quad L(M') \cap L(M'') \neq \Sigma^* & \text{(construction of } M', M''\text{)} \\ & \text{iff} \quad \mathsf{enc}(M', M'') \in L & \text{(definition of } L\text{).} \end{array}$$

- **Example 3**:

[**9 points**] Consider the following property of the RE languages defined over the alphabet $\Sigma = \{0, 1\}$:

$$\mathcal{P} \quad = \quad \{L \mid L \in \text{RE}, \quad \text{every string in } L \text{ has prefix } 111\}$$

and define $L_{\mathcal{P}} = \{\mathsf{enc}(M) \mid L(M) \in \mathcal{P}\}$.

(a) State whether $\emptyset \in \mathcal{P}$ is true or false, and motivate your answer.

(b) Use Rice's theorem to show that $L_{\mathcal{P}}$ is not in REC.

(c) Prove that $L_{\mathcal{P}}$ is not in RE.

(d) Consider the language

$$L_{\subseteq} \quad = \quad \{\mathsf{enc}(M_1, M_2) \mid L(M_1) \subseteq L(M_2)\} \, .$$

Specify a reduction $L_{\mathcal{P}} \leq_m L_{\subseteq}$ to show that $L_{\subseteq}$ is not in RE.

Solution:

- Lp不是re, Lp<m L, enc(M1, M2) ∈ Lp
- Iff definition
  - Lp definition: L(M) ∈ Lp
  - property: L(M) ⊆ {111} · Σ*
  - reduction: L(M1) ⊆ L(M2)
  - L: enc(M1, M2) ∈ L

$$\begin{array}{lll} \mathsf{enc}(M) \in L_{\mathcal{P}} & \text{iff} \quad L(M) \in \mathcal{P} & \text{(definition of } L_{\mathcal{P}}\text{)} \\ & \text{iff} \quad L(M) \subseteq \{111\} \cdot \Sigma^* & \text{(definition of } \mathcal{P}\text{)} \\ & \text{iff} \quad L(M_1) \subseteq L(M_2) & \text{(definition of our mapping)} \\ & \text{iff} \quad \mathsf{enc}(M_1, M_2) \in L_{\subseteq} & \text{(definition of } L_{\subseteq}\text{)} \end{array}$$

- **Example 4**:

  [8 points] In relation to the theory of Turing machines (TMs), answer the following questions. All the TMs introduced below are defined over the input alphabet $\Sigma = \{0, 1\}$.

  For a string $w \in \Sigma^*$ and a symbol $X \in \Sigma$, we write $\#_X(w)$ to represent the number of occurrences of X in w. We define $L_c = \{w \mid w \in \Sigma^*,\ \#_0(w) = \#_1(w)\}$. Consider the following property of the RE languages

  $$\mathcal{P} \quad = \quad \{L \mid L \in \text{RE},\ L \subseteq L_c\}$$

  and define $L_{\mathcal{P}} = \{\text{enc}(M) \mid L(M) \in \mathcal{P}\}$.

  (a) Use Rice's theorem to prove that $L_{\mathcal{P}}$ is not in REC.

  (b) Prove that $L_{\mathcal{P}}$ is not in RE.

  (c) For TMs M_1, M_2 and M_3, we write $\text{enc}(M_1, M_2, M_3)$ to represent some fixed binary encoding of these machines. Consider the language

  $$L \quad = \quad \{\text{enc}(M_1, M_2, M_3) \mid L(M_1) \subseteq L(M_2) \subseteq L(M_3)\}\ .$$

  Show that L is not in RE by establishing a reduction $L_{\mathcal{P}} \leq_m L$.

  Solution:

  - Lp不是re, Lp<m L, enc(M1, M2, M3) ∈ Lp
  - Iff definition
    - Lp definition: L(M) ⊆ Lc
    - property: ∅ ⊆ L(M) ⊆ Lc
    - reduction: L(M1) ⊆ L(M2) ⊆ L(M3)
    - L: enc(M1, M2, M3) ∈ L

  $$\begin{array}{lll} \text{enc}(M) \in L_{\mathcal{P}} & \text{iff} \quad L(M) \in \mathcal{P} & (\text{definition of } L_{\mathcal{P}}) \\ & \text{iff} \quad L(M) \subseteq L_c & (\text{definition of } \mathcal{P}) \\ & \text{iff} \quad \emptyset \subseteq L(M) \subseteq L_c & (\text{from set theory}) \\ & \text{iff} \quad L(M_1) \subseteq L(M_2) \subseteq L(M_3) & (\text{definition of our reduction}) \\ & \text{iff} \quad \text{enc}(M_1, M_2, M_3) \in L & (\text{definition of } L)\ . \end{array}$$

Mutual Induction

Closure properties of regular languages

- **If L is regular, Closure under complement**.

Theorem If L is a regular language over Σ, then so is $\overline{L} = \Sigma^* \setminus L$

Proof Let L be recognized by a DFA

$$A = (Q, \Sigma, \delta, q_0, F).$$

Let $B = (Q, \Sigma, \delta, q_0, Q \setminus F)$. Now $L(B) = \overline{L}$ □

- **If L is regular, Kleene closure**.

- Using structural induction, prove that a regular expression E over □ without the Kleene operation is also Regular language.

  (b) Base case: $E = \epsilon$, $E = \emptyset$, or $E = \boldsymbol{a}$ for $a \in \Sigma$. These regular expressions do not have any occurrence of the Kleene operator and generate a finite language by definition.

  Induction: Let E be a regular expression with no occurrence of the Kleene operator. According to the definition of regular expression in (a), we distinguish the following cases.

  i. If $E = F + G$, then F and G do not have any occurrence of the Kleene operator. We can apply the inductive hypothesis, deriving that $L(F)$ and $L(G)$ are both finite. Since $L(E) = L(F) \cup L(G)$, we have $|L(E)| \leq |L(F)| + |L(G)|$ and thus $L(E)$ is finite as well.

  ii. If $E = FG$, then F and G do not have any occurrence of the Kleene operator. We can apply the inductive hypothesis, deriving that $L(F)$ and $L(G)$ are both finite. Since $L(E) = L(F)L(G)$, we have $|L(E)| \leq |L(F)| \cdot |L(G)|$ and thus $L(E)$ is finite.

  iii. The case $E = F^*$ is not considered here, because E contains occurrence of the Kleene operator.

  iv. If $E = (F)$, then F does not have any occurrence of the Kleene operator. We can apply the inductive hypothesis, deriving that $L(F)$ is finite. Since $L(E) = L(F)$, $L(E)$ must be finite.

- **If L is regular, intersection closure**.

  Theorem If L and M are regular, then so is $L \cap M$

  Proof By De Morgan's law, $L \cap M = \overline{\overline{L} \cup \overline{M}}$

  We already know that regular languages are closed under complement and union

- **If L is regular, Closure under reverse operator**.
  - We proceed by struxctural induction on E.
  - **Base**: If E is ϵ, H, or a, then ER " E (easy to verify).

    Induction

    - $E = F + G$: We need to reverse the two languages. Then $E^R = F^R + G^R$
    - $E = F.G$: We need to reverse the two languages and also reverse the order of their concatenation. Then $E^R = G^R.F^R$
    - $E = F^*$:
      $w \in L(F^*)$ means $\exists k : w = w_1 w_2 \cdots w_k,\ w_i \in L(F)$
      then $w^R = w_k^R w_{k-1}^R \cdots w_1^R,\ w_i^R \in L(F^R)$
      then $w^R \in L(F^R)^*$
      Same reasoning for the inverse direction. Then $E^R = (F^R)^*$

    Thus $L(E^R) = (L(E))^R$ □

Closure properties of CFL

- **Regular language is CFL**: 3+4 cases for induction.

  Let E be any regular expression. We use a variable for E (start symbol) and a variable for each subexpression of E

  We use **structural induction** on the regular expression to build the productions of our CFG

  - if $E = \boldsymbol{a}$, then add production $E \to a$
  - if $E = \epsilon$, then add production $E \to \epsilon$
  - if $E = \varnothing$, then production set is empty
  - if $E = F + G$, then add production $E \to F \mid G$
  - if $E = FG$, then add production $E \to FG$
  - if $E = F^*$, then add production $E \to FE \mid \epsilon$
  - if $E = (F)$, then add production $E \to F$

---

5. Difficult parts

---

1. not TM but still

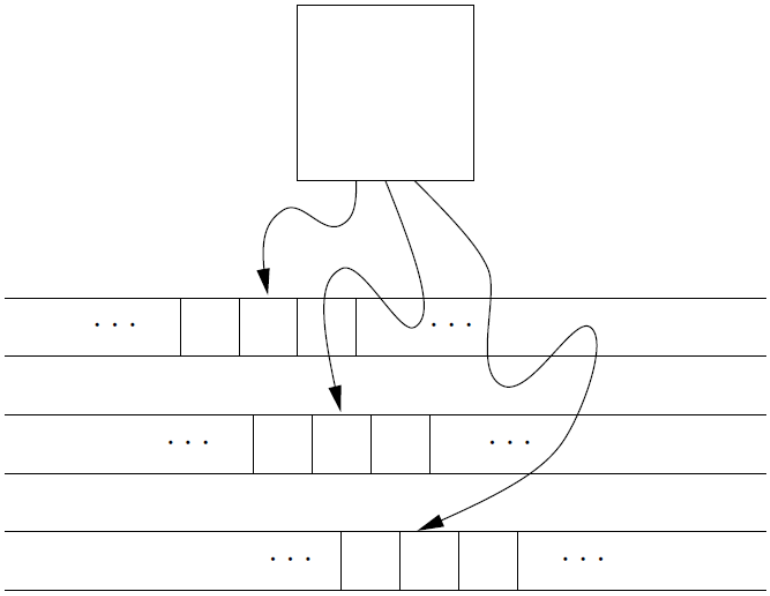
- **Conversion complexity**:
  - From NFA to DFA : computation takes exponential time
  - From DFA to NFA : put set brackets around the states computation takes time O(n), that is, linear time
- **Derivation of CFL**:
  - Leftmost derivation
  - Rightmost derivation
  - Parse tree

2. TM

- **Multi-tape TM**:

Multi-tape TM

We use a finite number of **independent** tapes for the computation, with the input on the first tape



In a single move the multi-tape TM performs the following actions

- state update, on the basis of read tape symbols
- for each tape :
  - write a symbol in current cell
  - move the tape head independently of the other heads (L = left, R = right, or S = stay)

Note that the stay option is not available in a TM

Theorem： A language accepted by a multi-tape TM M is RE

Theorem The TM N in the proof of the previous theorem simulates the first n moves of the TM M with k tapes in time $\mathcal{O}(n^2)$

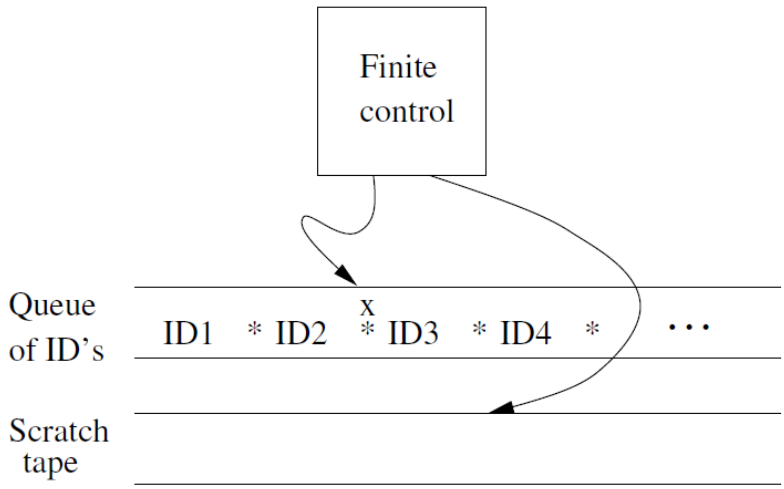
Proof (sketch) After n moves of M, tape head markers in N have **mutual distance** not exceeding $2n$

It follows that any one of the first n moves of M can be simulated by N in a number of moves not exceeding $4n + 2k$, which amounts to $\mathcal{O}(n)$ since k is a constant □

- **Nondeterministic TM**:

Theorem For each NTM M_N, there exists a (deterministic) TM M_D such that $L(M_N) = L(M_D)$

Proof (skecth) We specify M_D as a TM with two tapes



- **TM with semi-infinite tape**:

  Theorem Each language accepted by a TM M_2 is also accepted by a TM M_1 with semi-infinite tape

  Proof (sketch) First, we modify M_2 in such a way that it uses a **new** tape symbol B' each time B is used to overwrite a tape symbol

  Let $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, B, F_2)$ be the modified TM. We define

  $$M_1 = (Q_1, \Sigma \times \{B\}, \Gamma_1, \delta_1, q_0, [B, B], F_1)$$

- **Multi-Stack machine**:

  M is called a **multi-stack machine**, and can be viewed as a generalization of the deterministic PDA