Course Title: Formal Languages and Automata

UNIT 2:Regular Expressions

Mrs. Padmavati Sarode

S.Y B.Tech Div A and B

Regular Expressions

Definitions

Equivalence to Finite Automata

RE’s: Introduction

- *Regular expressions* are an algebraic way to describe languages.
- They describe exactly the regular languages.
- If E is a regular expression, then L(E) is the language it defines.
- We’ll describe RE’s and their languages recursively.

RE's: Definition

- Basis 1: If *a* is any symbol, then **a** is a RE, and L(**a**) = {a}.
  - Note: {a} is the language containing one string, and that string is of length 1.
- Basis 2: ε is a RE, and L(ε) = {ε}.
- Basis 3: ∅ is a RE, and L(∅) = ∅.

RE's: Definition – (2)

- Induction 1: If E_1 and E_2 are regular expressions, then E_1+E_2 is a regular expression, and $L(E_1+E_2) = L(E_1) \cup L(E_2)$.
- Induction 2: If E_1 and E_2 are regular expressions, then E_1E_2 is a regular expression, and $L(E_1E_2) = L(E_1)L(E_2)$.

Concatenation : the set of strings wx such that w
Is in $L(E_1)$ and x is in $L(E_2)$.L(E1)=001 and L(E2)=110
L(E1E2)=001110

RE's: Definition – (3)

- Induction 3: If E is a RE, then E* is a RE, and L(E*) = (L(E))*.

Closure, or "Kleene closure" = set of strings $w_1w_2\ldots w_n$, for some $n \geq 0$, where each w_i is in L(E).
Note: when n=0, the string is ε.

Precedence of Operators

- Parentheses may be used wherever needed to influence the grouping of operators.
- Order of precedence is * (highest), then concatenation, then + (lowest).

Examples: RE's

- L(**01**) = {01}.
- L(**01**+**0**) = {01, 0}.
- L(**0**(**1**+**0**)) = {01, 00}.
  - Note order of precedence of operators.
- L(**0***) = {ϵ, 0, 00, 000,… }.
- L((**0**+**10**)*(ϵ+**1**)) = all strings of 0's and 1's without two consecutive 1's.

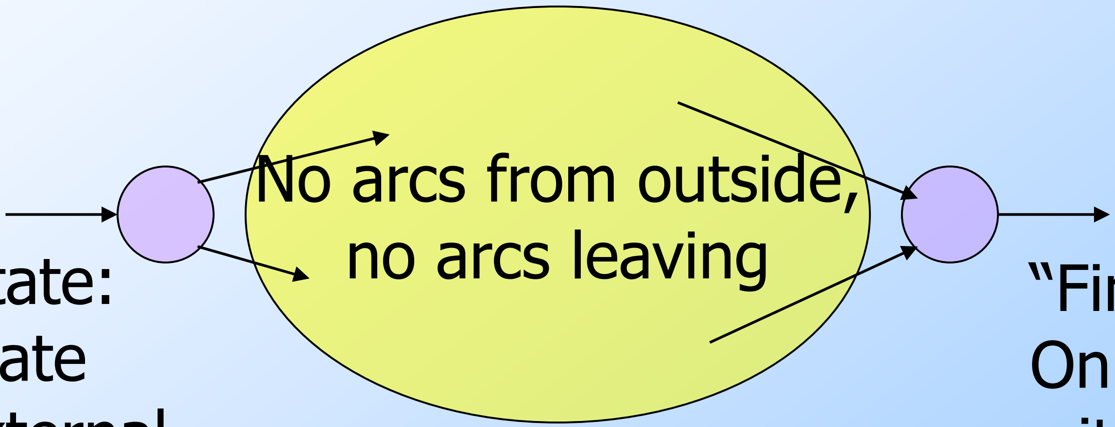
Equivalence of RE's and Automata

- We need to show that for every RE, there is an automaton that accepts the same language.
  - Pick the most powerful automaton type: the ϵ-NFA.
- And we need to show that for every automaton, there is a RE defining its language.
  - Pick the most restrictive type: the DFA.

Converting a RE to an ϵ-NFA

- Proof is an induction on the number of operators (+, concatenation, *) in the RE.
- We always construct an automaton of a special form (next slide).

Form of ϵ-NFA's Constructed

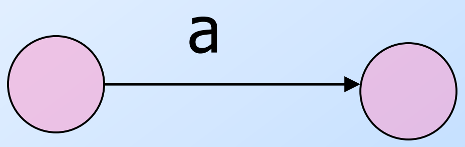


Start state:
Only state
with external
predecessors

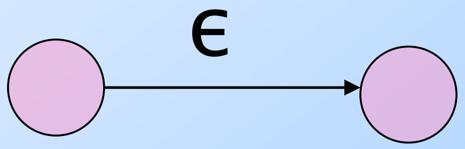
"Final" state:
Only state
with external
successors

RE to ϵ-NFA: Basis

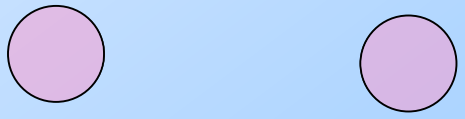
- Symbol **a**:



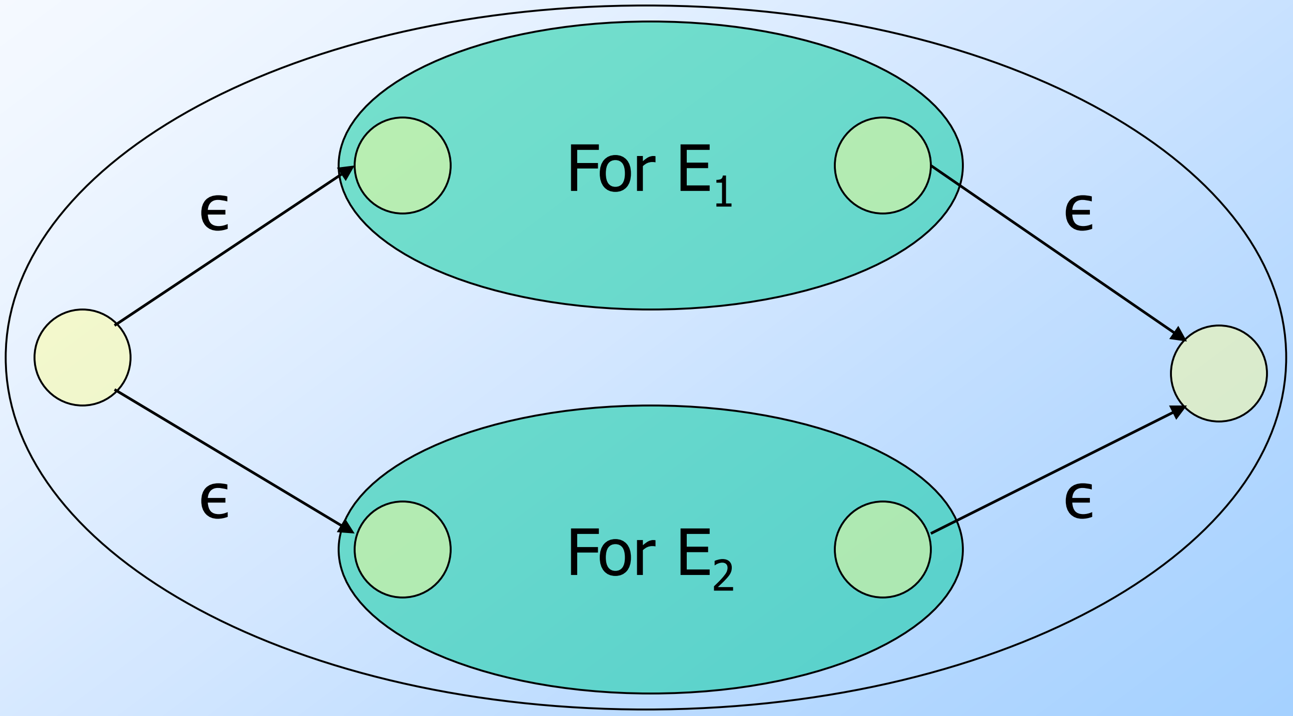
- ϵ:



- ∅:

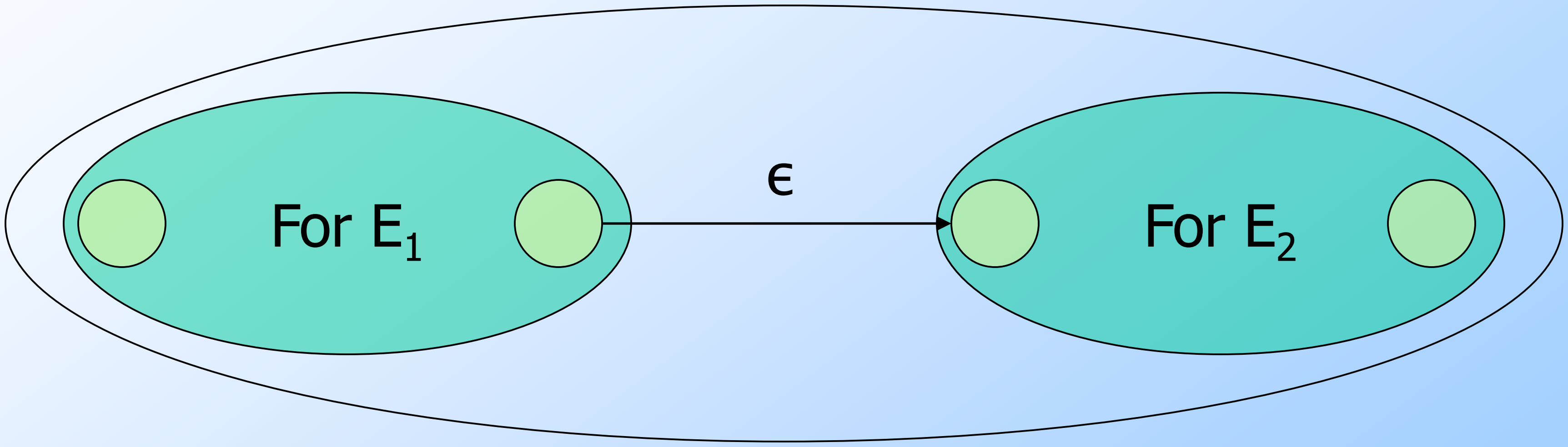


RE to ϵ-NFA: Induction 1 – Union



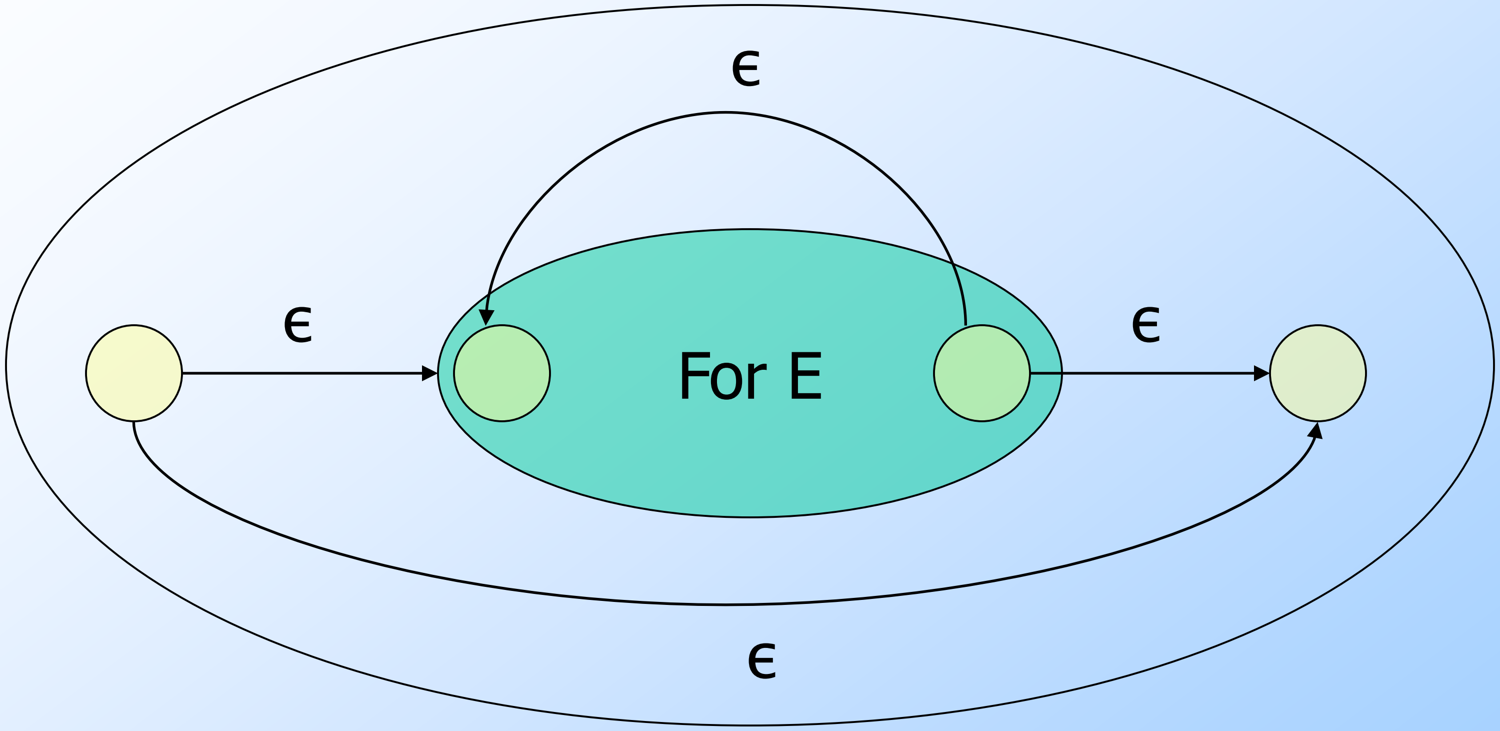
For $E_1 \cup E_2$

RE to ϵ-NFA: Induction 2 – Concatenation



For E_1E_2

RE to ϵ-NFA: Induction 3 – Closure



For E*

Algebraic Laws for RE's

- Union and concatenation behave sort of like addition and multiplication.
  - + is commutative and associative; concatenation is associative.
  - Commutative ex:4+5=9 ,5+4=9
    - A+B or B+A it gives same result in case of Regular expression if L(E1)+L(E2) here if you change order then it will not affect result.
    - Associative: if (2+4)+5=2+(4+5)=11,in case of RE if (a+b)+c=a+(b+c)
  - Concatenation distributes over +.
  - Exception: Concatenation is not commutative.

Applications of Regular Expressions

- **Applications:**
- Regular expressions are useful in a wide variety of text processing tasks, and more generally string processing, where the data need not be textual. Common applications include data validation, data scraping (especially web scraping), data wrangling, simple parsing, the production of syntax highlighting systems, and many other tasks.
- While regular expression's would be useful on Internet search engines, processing them across the entire database could consume excessive computer resources depending on the complexity and design of the regular expression's.

Algebraic Laws for Regular Expressions RegEx

- Associativity Laws for Regular Expressions RegEx

Let us see the Associativity Laws for Regular Expressions RegEx.

A + (B + C) = (A + B) + C and A.(B.C) = (A.B).C.

- Commutativity for Regular Expressions RegEx

Let us see the Commutativity for Regular Expressions RegEx.

A + B = B + A. However, A.B = B.A in general.

- Identity for Regular Expressions RegEx

Let us see the Identity for Regular Expressions RegEx.

∅ + A = A + ∅ = A and ε.A = A.ε = A

- Annihilator for Regular Expressions RegEx

Let us see the Annihilator for Regular Expressions RegEx.

∅.A = A.∅ = ∅

Continued…

- Distributivity for Regular Expressions RegEx

Let us see the Distributivity for Regular Expressions RegEx.

Left distributivity: A.(B + C) = A.B + A.C.

Right distributivity: (B + C).A = B.A + C.A.

Idempotent A + A = A.

- Closure Laws for Regular Expressions RegEx

Let us see the Closure Laws for Regular Expressions RegEx..

(A*)* = A*, ∅* = ε, ε* = ε, A+ = AA* = A*A, and A* = A + + ε.

- DeMorgan Type Law for Regular Expressions RegEx

Let us see the DeMorgan Type Law for Regular Expressions RegEx.

(L + B)* = (L*B*)*

Pumping Lemma for Regular Languages

- **Pumping Lemma for Regular Languages**
- The language accepted by the finite automata is called **Regular Language**. If we are given a language **L** and asked whether it is regular or not? So, to prove a given Language L is not regular we use a method called **Pumping Lemma.**
- The term **Pumping Lemma** is made up of two words:
- **Pumping:** The word pumping refers to generate many input strings by pushing a symbol in an input string again and again.
- **Lemma:** The word Lemma refers to intermediate theorem in a proof.

Continued….

Pumping Lemma is used to prove that given language is not regular. So, first of all we need to know when a language is called regular.

A language is called regular if:

Language is accepted by finite automata.

A regular grammar can be constructed to exactly generate the strings in a language.

A regular expression can be constructed to exactly generate the strings in a language.

Principle of Pumping Lemma

The pumping lemma states that all the regular languages have some special properties. If we can prove that the given language does not have those properties, then we can say that it is not a regular language.

Theorem 1: Pumping Lemma for Regular Languages

If L is an infinite regular language then there exists some positive integer n (pumping length) such that any string w ? L has length greater than or equal to n. i.e. **|w| >=n**, then string can be divided into three parts, w=xyz satisfying the following condition:

|w| >=n

For each i>=0, xy^iz ? L.

- |y| > 0
- |xy| <= n

|w| represents the length of string w and **y^i** means that i copies of y are concatenated together.

In the theory of formal languages, the pumping lemma for regular languages is a lemma that describes an essential property of all regular languages. Informally, it says that all sufficiently long strings in a regular language may be *pumped*—that is, have a middle section of the string repeated an arbitrary number of times—to produce a new string that is also part of the language.

Specifically, the pumping lemma says that for any regular language L there exists a constant p such that any string w in L with length at least p can be split into three substrings x, y and z (w=xyz with y being non-empty), such that the strings xz, xyz,xyyz,xyyyz,.. constructed by repeating Y zero or more times are still in L. This process of repetition is known as "pumping". Moreover, the pumping lemma guarantees that the length of xy will be at most p, imposing a limit on the ways in which w may be split. Finite languages vacuously satisfy the pumping lemma by having p equal to the maximum string length in L plus one.

Method to prove that a language L is not regular

- At first, we have to assume that L is regular.
- So, the pumping lemma should hold for L.
- Use the pumping lemma to obtain a contradiction − Select w such that

1. $|w| \geq n$.
2. Select y such that $|y| > 0$.
3. Select x such that $|xy| \leq n$. Assign the remaining string to z.

Example on pumping lemma

Using pumping lemma prove that the language A={a^n b^n|n>/0}

$A=\{a^n b^n \mid n \geq 0\}$

Assume that A is Regular

pumping length=p

$|y|>0$ ✓

$S=a^p b^p$

$$\begin{tikzcd} & S \arrow[dl] \arrow[d] \arrow[dr] & \\ x & y & z \end{tikzcd}$$

suppose pumping length p=7 S=aaaaaaabbbbbbb

$|xy|=6<p=7$

✓Case1 **:The y is in the a part** $S=\underbrace{aa}_{x}\underbrace{aaaa}_{y}\underbrace{abbbbbbb}_{z}$

Case2: **The y part is in the 'b' part** $S=\underbrace{aaaaaaabb}_{x}\underbrace{bbbb}_{y}\underbrace{b}_{z}$

✓ $|xy|<p$

$13\leq p$ ✗

✓Case3: **The y is in the a as well as 'b' part also**

$S=\underbrace{aaaaa}_{x}\underbrace{aabb}_{y}\underbrace{bbbbb}_{z}$

$|xy|=9\leq p$ ✗

By Rule $xy^iz \notin A$

case 1:xy^iz here i=2 aa aaaa aaaa abbbbbbb a's=11 and b's=7 a is not equal to b

case2:xy^z here i=2 s=aaaaaaabb bbbb bbbb b here also number of a's not equal to number of b's a=7 b=11

case 3: xy^iz i=2 S aaaaa aabb aabb bbbbb if we count a and b so again number of a's not equal to number of b's

S cannot be pumped==CONTRADICTICTION

we assumed that A is RL but S cannot be pumed so it is not RL

Pumping Lemma :If L is an infine language then there consits some positive integer 'n'(pumping length)such that any string w belongs to L has length greater than equal to |w|>=n then w can be divided into three parts w=xyz ,it should satisfy given conditions

1)for each i>=0 xy^i belongs to language

2)|y|>0

3)|xy|<=n

$$\begin{tikzcd}
L \arrow[r] & PLT \arrow[r, "Pass"] \arrow[d, "Fail"] & \textbf{undecidable} \\
& \textbf{Decidable i.e language is not regular} &
\end{tikzcd}$$

Ex Here $a^{n}b^{2n} \geqslant 0$

Take any string $\underbrace{aa}_{x}\underbrace{bb}_{y}\underbrace{bb}_{z} \in L$

Now Pump y value $xy^{i}z$

$\underbrace{aa}_{x}\underbrace{bbbb}_{y}\underbrace{bb}_{z} \notin L$

Let's's chk

$\underbrace{a}_{x}\underbrace{ab}_{y}\underbrace{bbb}_{z} \in L$ Now Pump y value $xy^{i}z$

$\underbrace{a}_{x}\underbrace{abab}_{y}\underbrace{bbb}_{z} \notin L$ **Hence it is not regular language**

Closure properties of Regular languages

Closure properties on regular languages are defined as certain operations on regular language which are guaranteed to produce regular language. Closure refers to some operation on a language, resulting in a new language that is of same “type” as originally operated on i.e., regular.
Regular languages are closed under following operations.
Consider L and M are regular languages:

1.Kleen Closure:
RS is a regular expression whose language is L, M. R* is a regular expression whose language is L*.

2.Positive closure:
RS is a regular expression whose language is L, M.R+ is a regular expression whose language is L+.

3. **Complement:**

   The complement of a language L (with respect to an alphabet E such that E^* contains L) is E^*–L. Since E^* is surely regular, the complement of a regular language is always regular.

4. **Reverse Operator:**

   Given language L, L^R is the set of strings whose reversal is in L.

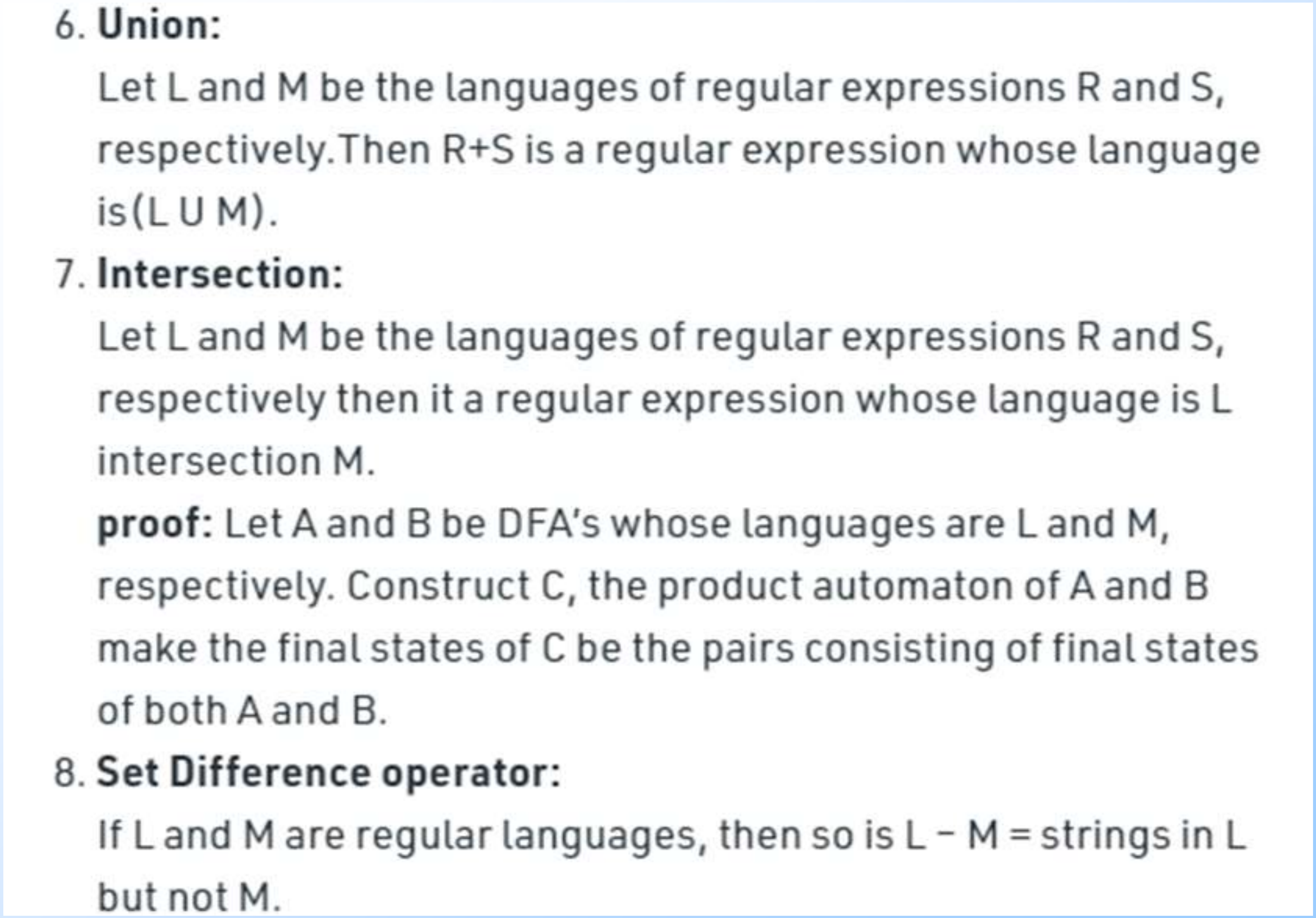
   Example: L = {0, 01, 100};

   L^R ={0, 10, 001}.

   Proof: Let E be a regular expression for L. We show how to reverse E, to provide a regular expression E^R for L^R.

5. **Complement:**

   The complement of a language L (with respect to an alphabet E such that E^* contains L) is E^*–L. Since E^* is surely regular, the complement of a regular language is always regular.



6. **Union:**

   Let L and M be the languages of regular expressions R and S, respectively.Then R+S is a regular expression whose language is(L U M).

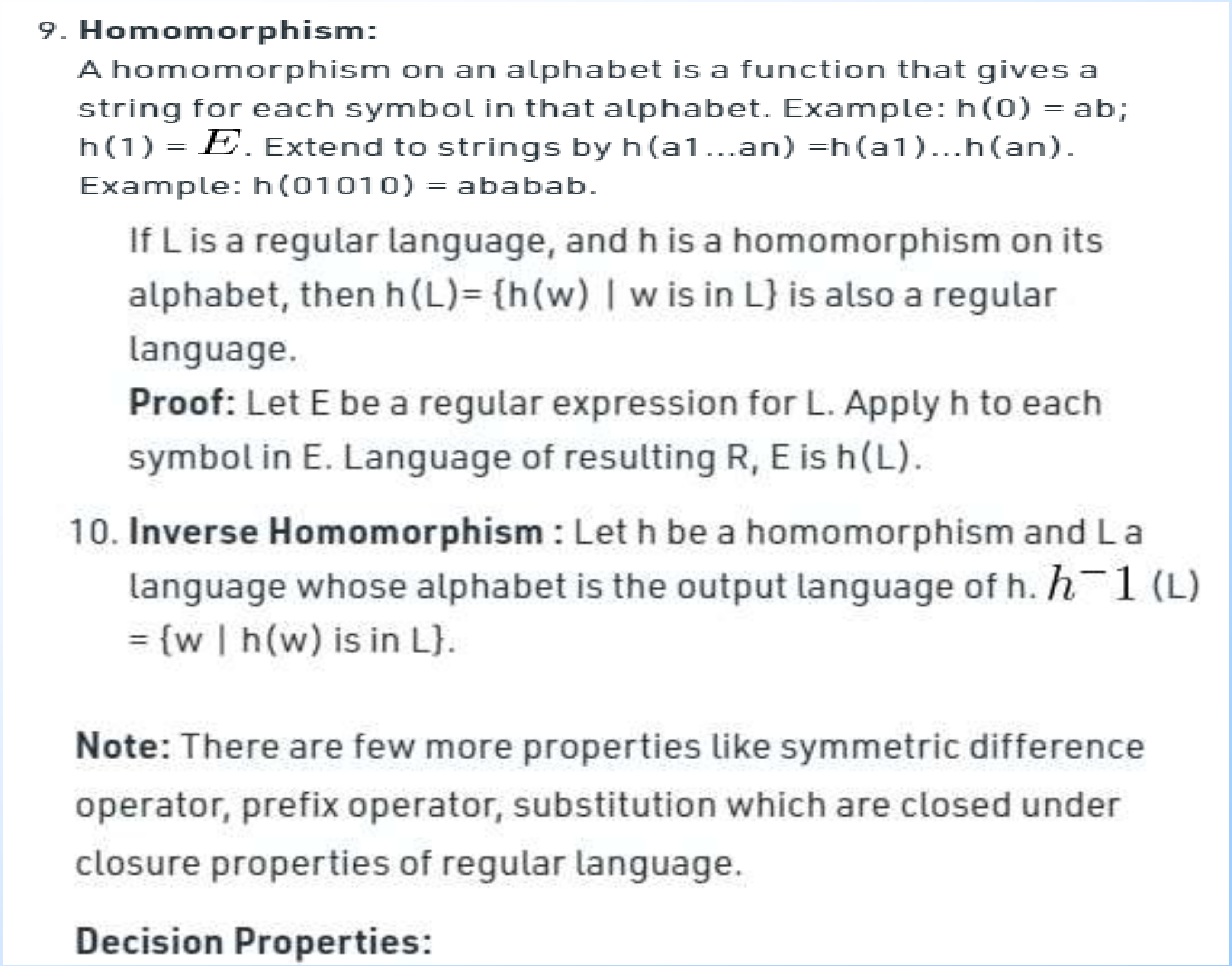
7. **Intersection:**

   Let L and M be the languages of regular expressions R and S, respectively then it a regular expression whose language is L intersection M.

   proof: Let A and B be DFA's whose languages are L and M, respectively. Construct C, the product automaton of A and B make the final states of C be the pairs consisting of final states of both A and B.

8. **Set Difference operator:**

   If L and M are regular languages, then so is L – M = strings in L but not M.



9. **Homomorphism:**
A homomorphism on an alphabet is a function that gives a string for each symbol in that alphabet. Example: h(0) = ab; h(1) = E. Extend to strings by h(a1...an) =h(a1)...h(an). Example: h(01010) = ababab.

If L is a regular language, and h is a homomorphism on its alphabet, then h(L)= {h(w) | w is in L} is also a regular language.
Proof: Let E be a regular expression for L. Apply h to each symbol in E. Language of resulting R, E is h(L).

10. **Inverse Homomorphism :** Let h be a homomorphism and L a language whose alphabet is the output language of h. $h^{-}1$ (L) = {w | h(w) is in L}.

Note: There are few more properties like symmetric difference operator, prefix operator, substitution which are closed under closure properties of regular language.

Decision Properties:

Decision Properties:

Approximately all the properties are decidable in case of finite automaton.

(i) Emptiness
(ii) Non-emptiness
(iii) Finiteness
(iv) Infiniteness
(v) Membership
(vi) Equality

These are explained as following below.

(i) Emptiness and Non-emptiness:

Step-1: select the state that cannot be reached from the initial states & delete them (remove unreachable states).

Step 2: if the resulting machine contains at least one final states, so then the finite automata accepts the non-empty language.

Step 3: if the resulting machine is free from final state, then finite automata accepts empty language.

(ii) Finiteness and Infiniteness:

Step-1: select the state that cannot be reached from the initial state & delete them (remove unreachable states).

Step-2: select the state from which we cannot reach the final state & delete them (remove dead states).

Step-3: if the resulting machine contains loops or cycles then the finite automata accepts infinite language.

Step-4: if the resulting machine do not contain loops or cycles then the finite automata accepts infinite language.

(iii) Membership:

Membership is a property to verify an arbitrary string is accepted by a finite automaton or not i.e. it is a member of the language or not.

Let M is a finite automata that accepts some strings over an alphabet, and let ‘w’ be any string defined over the alphabet, if there exist a transition path in M, which starts at initial state & ends in anyone of the final state, then string ‘w’ is a member of M, otherwise ‘w’ is not a member of M.

(iv) Equality:

Two finite state automata M1 & M2 is said to be equal if and only if, they accept the same language. Minimize the finite state automata and the minimal DFA will be unique.