Patna Women's College

MCA Department

Semester II

MCA CS2T07 Automata Theory Notes

Formal language

The alphabet of a formal language is the set of symbols, letters, or tokens from which the strings of the language may be formed; frequently it is required to be finite. The strings formed from this alphabet are called words, and the words that belong to a particular formal language are sometimes called well-formed words or well-formed formulas. A formal language is often defined by means of a formal grammar such as a regular grammar or context-free grammar, also called its formation rule.

The field of formal language theory studies the purely syntactical aspects of such languages— that is, their internal structural patterns. Formal language theory sprang out of linguistics, as a way of understanding the syntactic regularities of natural languages. In computer science, formal languages are often used as the basis for defining programming languages and other systems in which the words of the language are associated with particular meanings or semantics.

A formal language L over an alphabet Σ is a subset of Σ^*, that is, a set of words over that alphabet.

In computer science and mathematics, which do not usually deal with natural languages, the adjective "formal" is often omitted as redundant.

While formal language theory usually concerns itself with formal languages that are described by some syntactical rules, the actual definition of the concept "formal language" is only as above: a (possibly infinite) set of finite-length strings, no more nor less. In practice, there are many languages that can be described by rules, such as regular languages or context-free languages.

The notion of a formal grammar may be closer to the intuitive concept of a "language," one described by syntactic rules.

Formal language

A formal grammar (sometimes simply called a grammar) is a set of formation rules for strings in a formal language. The rules describe how to form strings from the language's alphabet that are

valid according to the language's syntax. A grammar does not describe the meaning of the strings or what can be done with them in whatever context—only their form.

A formal grammar is a set of rules for rewriting strings, along with a "start symbol" from which rewriting must start. Therefore, a grammar is usually thought of as a language generator.

However, it can also sometimes be used as the basis for a "recognizer"—a function in computing that determines whether a given string belongs to the language or is grammatically incorrect. To describe such recognizers, formal language theory uses separate formalisms, known as automata theory. One of the interesting results of automata theory is that it is not possible to design a recognizer for certain formal languages.

Alphabet

An alphabet, in the context of formal languages, can be any set, although it often makes sense to use an alphabet in the usual sense of the word, or more generally a character set such as ASCII. Alphabets can also be infinite; e.g. first-order logic is often expressed using an alphabet which, besides symbols such as∧, ¬, □ and parentheses, contains infinitely many elements x0, x1, x2, … that play the role of variables. The elements of an alphabet are called its letters.

word

A word over an alphabet can be any finite sequence, or string, of letters. The set of all words over an alphabet Σ is usually denoted by Σ* (using the Kleene star). For any alphabet there is only one word of length 0, the empty word, which is often denoted by e, ε or λ. By concatenation one can combine two words to form a new word, whose length is the sum of the lengths of the original words. The result of concatenating a word with the empty word is the original word.

Operations on languages

Certain operations on languages are common. This includes the standard set operations, such as union, intersection, and complement. Another class of operation is the element-wise application of string operations.

Examples: suppose L1 and L2 are languages over some common alphabet.

The concatenation L1L2 consists of all strings of the form vw where v is a string from L1 and w is a string from L2.

The intersection L1 ∩ L2 of L1 and L2 consists of all strings which are contained in both languages

The complement ¬L of a language with respect to a given alphabet consists of all strings over the alphabet that are not in the language.

The Kleene star: the language consisting of all words that are concatenations of 0 or more words in the original language;

Reversal:

Let e be the empty word, then eR = e, and

for each non-empty word w = x1…xn over some alphabet, let wR = xn…x1,

then for a formal language L, LR = {wR | w □ L}.

String homomorphism

Such string operations are used to investigate closure properties of classes of languages. A class of languages is closed under a particular operation when the operation, applied to languages in the class, always produces a language in the same class again. For instance, the context-free languages are known to be closed under union, concatenation, and intersection with regular languages, but not closed under intersection or complement. The theory of trios and abstract families of languages studies the most common closure properties of language families in their own right.

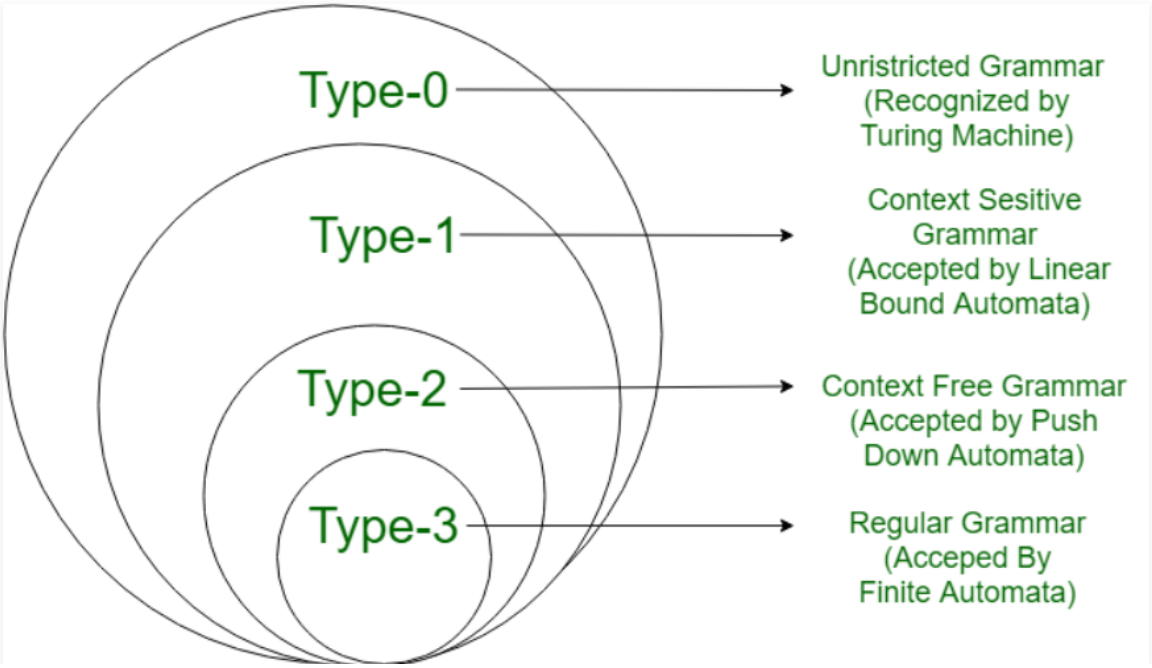
Language

“A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols.

Chomsky Hierarchy in Theory of Computation

According to Chomsky hierarchy, grammars are divided of 4 types:

```
Type 0 known as unrestricted grammar.
Type 1 known as context sensitive grammar.
Type 2 known as context free grammar.
Type 3 Regular Grammar.
```



Regular Expressions, Regular Grammar and Regular Languages

As discussed in Chomsky Hierarchy, Regular Languages are the most restricted types of languages and are accepted by finite automata.

Regular Expressions

Regular Expressions are used to denote regular languages. An expression is regular if:

- ϕ is a regular expression for regular language ϕ.
- ε is a regular expression for regular language $\{\varepsilon\}$.
- If $a \in \Sigma$ (Σ represents the input alphabet), a is regular expression with language {a}.
- If a and b are regular expression, a + b is also a regular expression with language {a,b}.
- If a and b are regular expression, ab (concatenation of a and b) is also regular.
- If a is regular expression, a* (0 or more times a) is also regular.

	REGULAR EXPRESSION	REGULAR LANGUAGES
set of vovels	(a ∪ e ∪ i ∪ o ∪ u)	{a, e, i, o, u}
a followed by 0 or more b	$(a.b^*)$	{a, ab, abb, abbb, abbbb,....}
any no. of vowels followed by any no. of consonants	$v^*.c^*$ (where v – vowels and c – consonants)	{ ε , a ,aou, aiou, b, abcd.....} where ε represent empty string (in case 0 vowels and o consonants)

Regular Grammar : A grammar is regular if it has rules of form A -> a or A -> aB or A -> ε where ε is a special symbol called NULL.

Regular Languages : A language is regular if it can be expressed in terms of regular expression.

Closure Properties of Regular Languages

Union : If L1 and If L2 are two regular languages, their union L1 ∪ L2 will also be regular. For example, $L1 = \{a^n \mid n \geq 0\}$ and $L2 = \{b^n \mid n \geq 0\}$

$L3 = L1 \cup L2 = \{a^n \cup b^n \mid n \geq 0\}$ is also regular.

Intersection : If L1 and If L2 are two regular languages, their intersection L1 ∩ L2 will also be regular. For example,

$L1 = \{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$ and $L2 = \{a^m b^n \cup b^n a^m \mid n \geq 0 \text{ and } m \geq 0\}$

$L3 = L1 \cap L2 = \{a^m b^n \mid n \geq 0 \text{ and } m \geq 0\}$ is also regular.

Concatenation : If L1 and If L2 are two regular languages, their concatenation L1.L2 will also be regular. For example,

$L1 = \{a^n \mid n \geq 0\}$ and $L2 = \{b^n \mid n \geq 0\}$

$L3 = L1.L2 = \{a^m . b^n \mid m \geq 0 \text{ and } n \geq 0\}$ is also regular.

Kleene Closure : If L1 is a regular language, its Kleene closure L1* will also be regular. For example,

$L1 = (a \cup b)$

$L1^* = (a \cup b)^*$

Complement : If L(G) is regular language, its complement L'(G) will also be regular. Complement of a language can be found by subtracting strings which are in L(G) from all possible strings. For example,

$L(G) = \{a^n \mid n > 3\}$

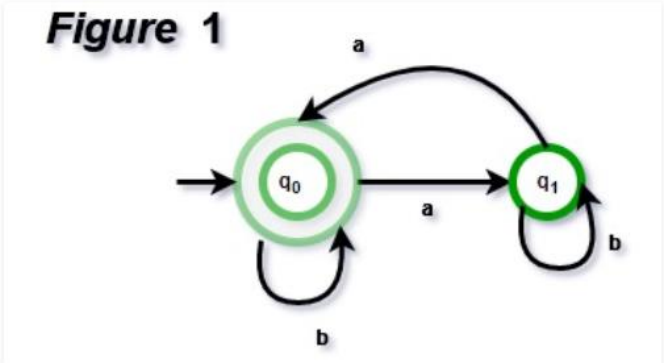
$L'(G) = \{a^n \mid n <= 3\}$

Note : Two regular expressions are equivalent if languages generated by them are same. For example, (a+b*)* and (a+b)* generate same language. Every string which is generated by (a+b*)* is also generated by (a+b)* and vice versa.

Designing Finite Automata from Regular Expression (Set 1)

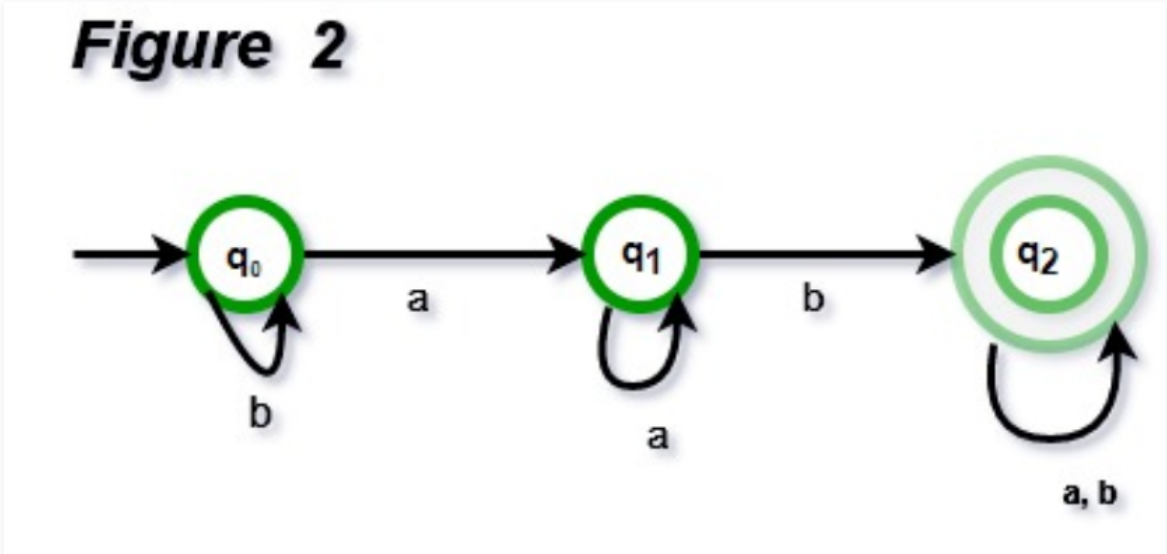
In this article, we will see some popular regular expressions and how we can convert them to finite automata.

- **Even number of a's :** The regular expression for even number of a's is **(b|ab*ab*)***. We can construct a finite automata as shown in Figure 1.



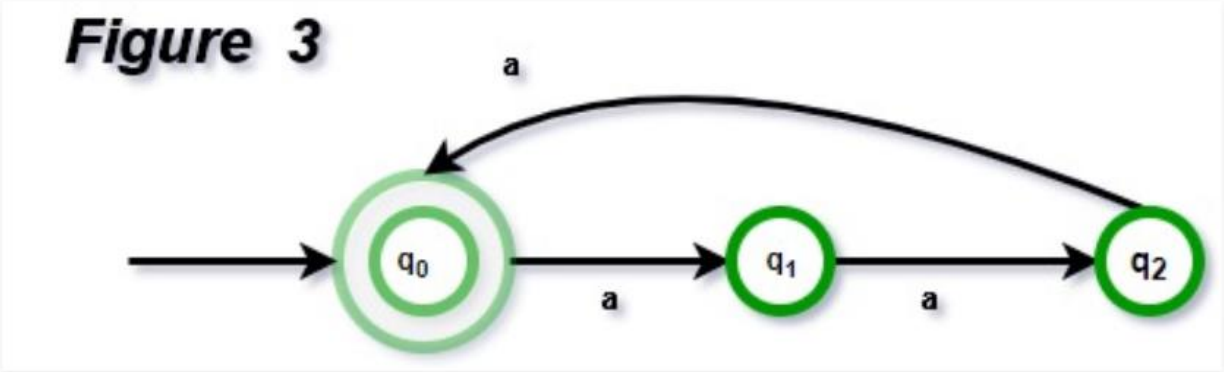
The above automata will accept all strings which have even number of a's. For zero a's, it will be in q0 which is final state. For one 'a', it will go from q0 to q1 and the string will not be accepted. For two a's at any positions, it will go from q0 to q1 for 1st 'a' and q1 to q0 for second 'a'. So, it will accept all strings with even number of a's.

- **String with 'ab' as substring :** The regular expression for strings with 'ab' as substring is **(a|b)*ab(a|b)***. We can construct finite automata as shown in Figure 2.



The above automata will accept all string which have ‘ab’ as substring. The automata will remain in initial state q0 for b’s. It will move to q1 after reading ‘a’ and remain in same state for all ‘a’ afterwards. Then it will move to q2 if ‘b’ is read. That means, the string has read ‘ab’ as substring if it reaches q2.

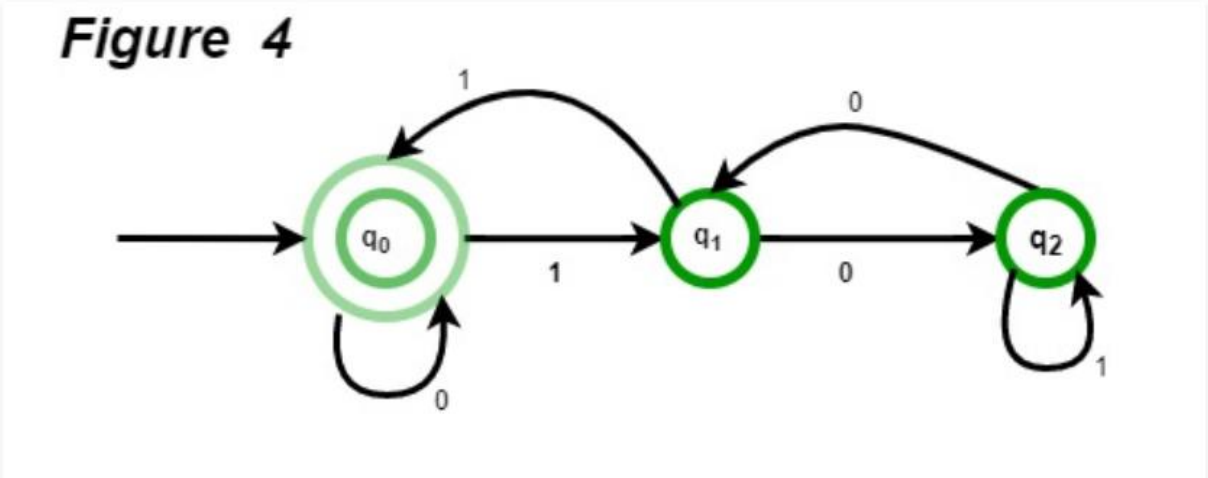
- **String with count of ‘a’ divisible by 3 :** The regular expression for strings with count of a divisible by 3 is $\{a^{3n} \mid n >= 0\}$. We can construct automata as shown in Figure 3.



The above automata will accept all string of form a^{3n}. The automata will remain in initial state q0 for ε and it will be accepted. For string ‘aaa’, it will move from q0 to q1 then q1 to q2 and then q2 to q0. For every set of three a’s, it will come to q0, hence accepted. Otherwise, it will be in q1 or q2, hence rejected.

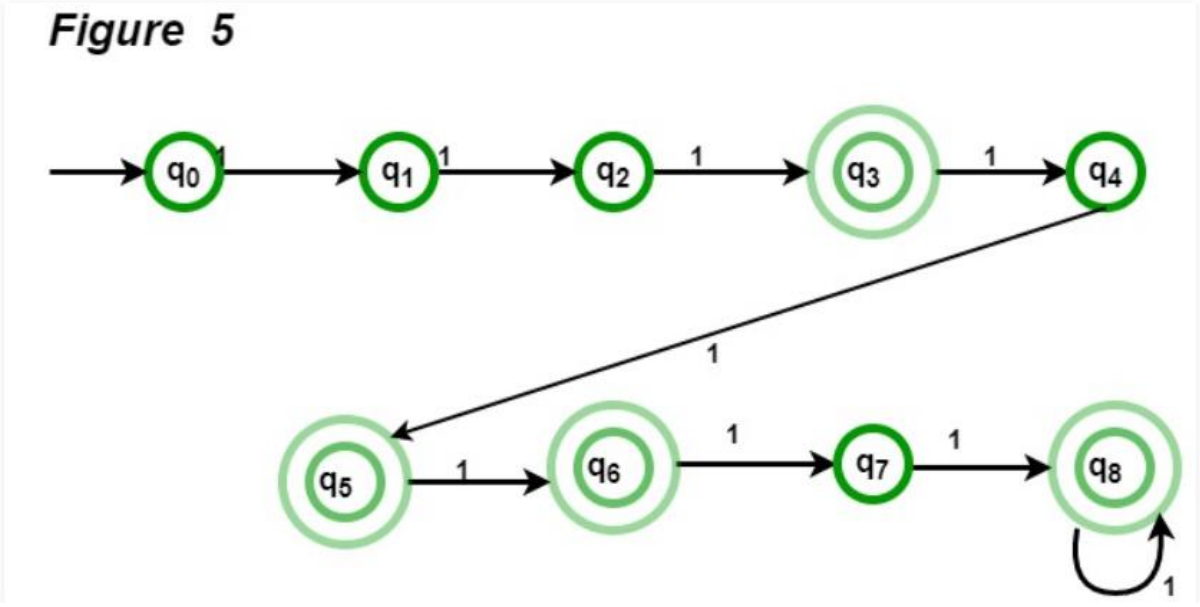
Note : If we want to design a finite automata with number of a's as 3n+1, same automata can be used with final state as q1 instead of q0.
If we want to design a finite automata with language $\{a^{kn} \mid n >= 0\}$, k states are required. We have used k = 3 in our example.

- **Binary numbers divisible by 3 :** The regular expression for binary numbers which are divisible by three is **(0|1(01*0)*1)***. The examples of binary number divisible by 3 are 0, 011, 110, 1001, 1100, 1111, 10010 etc. The DFA corresponding to binary number divisible by 3 can be shown in Figure 4.



The above automata will accept all binary numbers divisible by 3. For 1001, the automata will go from q0 to q1, then q1 to q2, then q2 to q1 and finally q2 to q0, hence accepted. For 0111, the automata will go from q0 to q0, then q0 to q1, then q1 to q0 and finally q0 to q1, hence rejected.

- **String with regular expression (111 + 11111)* :** The string accepted using this regular expression will have 3, 5, 6(111 twice), 8 (11111 once and 111 once), 9 (111 thrice), 10 (11111 twice) and all other counts of 1 afterwards. The DFA corresponding to given regular expression is given in Figure 5.



Identities of RE

1. $\emptyset + R = R + \emptyset = R$
2. $\emptyset \cdot R = R \cdot \emptyset = \emptyset$
3. $\epsilon \cdot R = R \cdot \epsilon = R$
4. $\epsilon^* = \epsilon$
5. $\emptyset^* = \epsilon$
6. $\epsilon + RR^* = R^*R + \epsilon = R^*$
7. $(a+b)^* = (a^* + b^*)^*$

$= (a^* b^*)^*$

$= (a^* + b)^*$

$= (a + b^*)^*$

$= a^*(ba^*)^*$

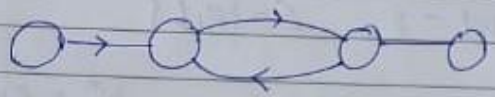
$= b^*(ab^*)^*$

Testing whether a given language is Regular or not.

① Finite language – Regular – RE,
we use + +

② Infinite language.

$L = \{ ab, abab, ababab, \ldots \}$

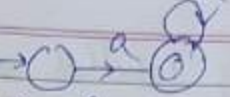


Pumping Lemma → If we doesnot find a pattern in an infinite language, that infinit language is not regular.

It is used to check that the language is NOT REGULAR.

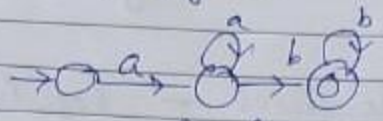
$L = \{ ab, abab, ababab, \ldots \} \cup \{ a^p / p \text{ is prime} \}$

① a^n / $n \geq 1$



∴ Regular

② $a^n b^m$ / $n, m \geq 1$



Regular.

③ $a^n b^n$ / $n \leq 10^{10^{10}}$ → Since it is bounded
Therefore, it is Regular.

④ $a^n b^n$ / $n \geq 1$ → Not Regular.
Finite automata have finite memory.
FA can't count this infinite counting.

⑤ ww^R / $|w| = 2$ $\Sigma = \{a, b\}$

ex = w = aa, ab, ba, bb

w^R = aa, ba, ab, bb

ww^R
aa aa
abba
baab
bbbb

Since, the language is finite, we can say that this is Regular.

⑥ ww^R / [redacted] $w \in (a, b)^*$
Finite Automata doesnot have the capacity to save infinite length string.

(7) $ww \,/\, w \in (a,b)^*$

ex - aabbaabb

— Not Regular.

(8) $a^n b^m c^k \,/\, n, m, k \geq 1$: $aa^*bb^*cc^*$

— Regular.

(9) $a^i b^{2j} \,/\, i, j \geq 1$

→ Regular

$aa^*bb(bb)^*$

(10) [redacted]

$a^i b^{4j} \,/\, i, j \geq 1$

→ Regular

(11) $\Sigma = \{a\}$

$a^n \,/\, n$ is even → Regular → $\{a^0, a^2, a^4, a^6, a^8, \ldots\}$

$a^n \,/\, n$ is odd → Regular → $\{a^1, a^3, a^5, a^7, \ldots\}$

$a^n \,/\, n$ is prime → Not Regular

$a^{n^2} \,/\, n \geq 1$ → Not Regular

$a^{2^n} \,/\, n \geq 1$ → Not Regular

→ $\{a^2, a^4, a^8, a^{16}, a^{32} \ldots\}$

$\Sigma = \{a, b\}$

(12) $a^i b^{j^2}$ / $i, j \geq 1$ ✗ Not Regular

(13) $a^i b^{2^n}$ ✗ Not Regular

(14) $a^i b^p$ / $i \geq 1$ & p is prime.

(15) $\Sigma = \{a, b\}$ $w \in (a, b)^*$

$\{ w / n_a(w) = n_b(w) \}$ infinite length.

→ Not Regular.

(16) $\{ w / n_a(w) \bmod 4 = n_b(w) \bmod 4 \}$

→ we can build its Finite Automata, That's why it is Regular.

(17) wwW^R / $w \in (a, b)^*$

– Not Regular

(18) $a^n b^n c^n$ / $n \geq 1$

– Not Regular

(19) $a^n b^{n+m} c^m$ / $n, m \geq 1$

→ Not Regular

References:

https://www.geeksforgeeks.org/chomsky-hierarchy-in-theory-of-computation/?ref=lbp

https://www.geeksforgeeks.org/regular-expressions-regular-grammar-and-regular-languages/

https://www.youtube.com/watch?v=eqCkkC9A0Q4&list=PLEbnTDJUr_IdM___FmDFBJBz0zCsOFxfK

REFERENCE BOOKS

Hopcroft, Ullman " Theory of Computation & Formal Languages", TMH.

FORMAL LANGUAGES AND AUTOMATA THEORY, H S Behera, Janmenjoy Nayak , Hadibandhu Pattnayak, Vikash Publishing, New Delhi.

Anand Sharma, "Theory of Automata and Formal Languages", Laxmi Publisher

Faculty:

Hera Shaheen

Assistant Professor

MCA Department

Patna Women's College