**NLP - Word Level Analysis**

## **Regular Expressions**

A regular expression (RE) is a language for specifying text search strings. RE helps us to match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are used to search texts in UNIX as well as in MS WORD in identical way. We have various search engines using a number of RE features.

Mathematically, A Regular Expression can be defined as follows −

ε is a Regular Expression, which indicates that the language is having an empty string.

φ is a Regular Expression which denotes that it is an empty language.

If X and Y are Regular Expressions, then

X, Y

X.Y(Concatenation of XY)

X+Y (Union of X and Y)

X\*, Y\* (Kleen Closure of X and Y)

|  |  |
| --- | --- |
| **Regular Expressions** | **Regular Set** |
| (0 + 10\*) | {0, 1, 10, 100, 1000, 10000, … } |
| (0\*10\*) | {1, 01, 10, 010, 0010, …} |
| (0 + ε)(1 + ε) | {ε, 0, 1, 01} |
| (a+b)\* | It would be set of strings of a’s and b’s of any length which also includes the null string i.e. {ε, a, b, aa , ab , bb , ba, aaa…….} |
| (a+b)\*abb | It would be set of strings of a’s and b’s ending with the string abb i.e. {abb, aabb, babb, aaabb, ababb, …………..} |
| (11)\* | It would be set consisting of even number of 1’s which also includes an empty string i.e. {ε, 11, 1111, 111111, ……….} |
| (aa)\*(bb)\*b | It would be set of strings consisting of even number of a’s followed by odd number of b’s i.e. {b, aab, aabbb, aabbbbb, aaaab, aaaabbb, …………..} |
| (aa + ab + ba + bb)\* | It would be string of a’s and b’s of even length that can be obtained by concatenating any combination of the strings aa, ab, ba and bb including null i.e. {aa, ab, ba, bb, aaab, aaba, …………..} |

## **Types of Finite State Automation (FSA)**

Finite state automation is of two types. Let us see what the types are.

### **Deterministic Finite automation (DFA)**

It may be defined as the type of finite automation wherein, for every input symbol we can determine the state to which the machine will move. It has a finite number of states that is why the machine is called Deterministic Finite Automaton (DFA).

Mathematically, a DFA can be represented by a 5-tuple (Q, Σ, δ, q0, F), where −

* Q is a finite set of states.
* Σ is a finite set of symbols, called the alphabet of the automaton.
* δ is the transition function where δ: Q × Σ → Q .
* q0 is the initial state from where any input is processed (q0 ∈ Q).
* F is a set of final state/states of Q (F ⊆ Q).

Whereas graphically, a DFA can be represented by diagraphs called state diagrams where −

* The states are represented by **vertices**.
* The transitions are shown by labeled **arcs**.
* The initial state is represented by an **empty incoming arc**.
* The final state is represented by **double circle**.

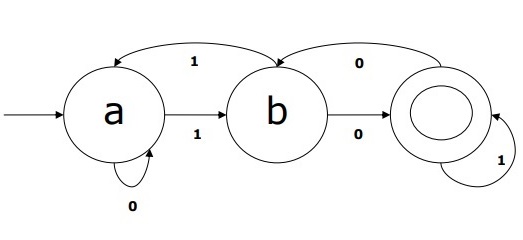
### **Example of DFA**

Suppose a DFA be

* Q = {a, b, c},
* Σ = {0, 1},
* q0 = {a},
* F = {c},
* Transition function δ is shown in the table as follows −

|  |  |  |
| --- | --- | --- |
| **Current State** | **Next State for Input 0** | **Next State for Input 1** |
| A | a | B |
| B | b | A |
| C | c | C |

The graphical representation of this DFA would be as follows −



### **Non-deterministic Finite Automation (NDFA)**

It may be defined as the type of finite automation where for every input symbol we cannot determine the state to which the machine will move i.e. the machine can move to any combination of the states. It has a finite number of states that is why the machine is called Non-deterministic Finite Automation (NDFA).

Mathematically, NDFA can be represented by a 5-tuple (Q, Σ, δ, q0, F), where −

* Q is a finite set of states.
* Σ is a finite set of symbols, called the alphabet of the automaton.
* δ :-is the transition function where δ: Q × Σ → 2 Q.
* q0 :-is the initial state from where any input is processed (q0 ∈ Q).
* F :-is a set of final state/states of Q (F ⊆ Q).

Whereas graphically (same as DFA), a NDFA can be represented by diagraphs called state diagrams where −

* The states are represented by **vertices**.
* The transitions are shown by labeled **arcs**.
* The initial state is represented by an **empty incoming arc**.
* The final state is represented by double **circle**.

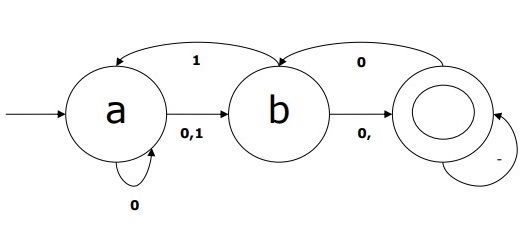
### **Example of NDFA**

Suppose a NDFA be

* Q = {a, b, c},
* Σ = {0, 1},
* q0 = {a},
* F = {c},
* Transition function δ is shown in the table as follows −

|  |  |  |
| --- | --- | --- |
| **Current State** | **Next State for Input 0** | **Next State for Input 1** |
| A | a, b | B |
| B | C | a, c |
| C | b, c | C |

The graphical representation of this NDFA would be as follows −



## **Morphological Parsing**

The term morphological parsing is related to the parsing of morphemes. We can define morphological parsing as the problem of recognizing that a word breaks down into smaller meaningful units called morphemes producing some sort of linguistic structure for it. For example, we can break the word *foxes* into two, *fox* and *-es*. We can see that the word *foxes*, is made up of two morphemes, one is *fox* and other is *-es*.

In other sense, we can say that morphology is the study of −

* The formation of words.
* The origin of the words.
* Grammatical forms of the words.
* Use of prefixes and suffixes in the formation of words.
* How parts-of-speech (PoS) of a language are formed.

## **Types of Morphemes**

Morphemes, the smallest meaning-bearing units, can be divided into two types −

* Stems
* Word Order

### **Stems**

It is the core meaningful unit of a word. We can also say that it is the root of the word. For example, in the word foxes, the stem is fox.

* **Affixes** − As the name suggests, they add some additional meaning and grammatical functions to the words. For example, in the word foxes, the affix is − es.

Further, affixes can also be divided into following four types −

* + **Prefixes** − As the name suggests, prefixes precede the stem. For example, in the word unbuckle, un is the prefix.
  + **Suffixes** − As the name suggests, suffixes follow the stem. For example, in the word cats, -s is the suffix.
  + **Infixes** − As the name suggests, infixes are inserted inside the stem. For example, the word cupful, can be pluralized as cupsful by using -s as the infix.
  + **Circumfixes** − They precede and follow the stem. There are very less examples of circumfixes in English language. A very common example is ‘A-ing’ where we can use -A precede and -ing follows the stem.

### **Word Order**

The order of the words would be decided by morphological parsing. Let us now see the requirements for building a morphological parser −

### **Lexicon**

The very first requirement for building a morphological parser is lexicon, which includes the list of stems and affixes along with the basic information about them. For example, the information like whether the stem is Noun stem or Verb stem, etc.

### **Morphotactics**

It is basically the model of morpheme ordering. In other sense, the model explaining which classes of morphemes can follow other classes of morphemes inside a word. For example, the morphotactic fact is that the English plural morpheme always follows the noun rather than preceding it.

**Orthographic rules**

These spelling rules are used to model the changes occurring in a word. For example, the rule of converting y to ie in word like city+s = cities not citys.

Word level analysis" in NLP, combined with "**unsmoothed n-grams**," refers to a method of analyzing text by examining individual words within a sequence, where the probability of each word appearing is calculated directly based on its frequency in the corpus, without any adjustments to account for rare words or data sparsity, essentially providing a raw measure of how often a specific word appears in a given context.

Key points about word level analysis with unsmoothed n-grams:

**N-gram concept:**

An "n-gram" is a sequence of "n" words appearing together in a text; a unigram (n=1) represents a single word, a bigram (n=2) represents a pair of words, and so on.

**Unsmoothed calculation:**

When using an unsmoothed n-gram model, the probability of a word appearing is simply calculated by dividing the count of that word (in the specific n-gram sequence) by the total count of all n-grams in the corpus.

**Data sparsity issue:**

The main drawback of unsmoothed n-grams is that rare words or combinations of words that don't appear frequently in the training data will have a probability of zero, which can lead to issues in applications like text generation or language modeling.

**Applications:**

Despite the data sparsity issue, unsmoothed n-grams can be useful in specific scenarios like:

Identifying key phrases: By analyzing the frequency of specific word combinations (bigrams, trigrams) in a text corpus, you can identify important phrases related to a topic.

**Basic language modeling:** For simple text generation tasks where the focus is on capturing common word sequences.

Feature engineering: N-grams can be used as features in machine learning models for text classification tasks.