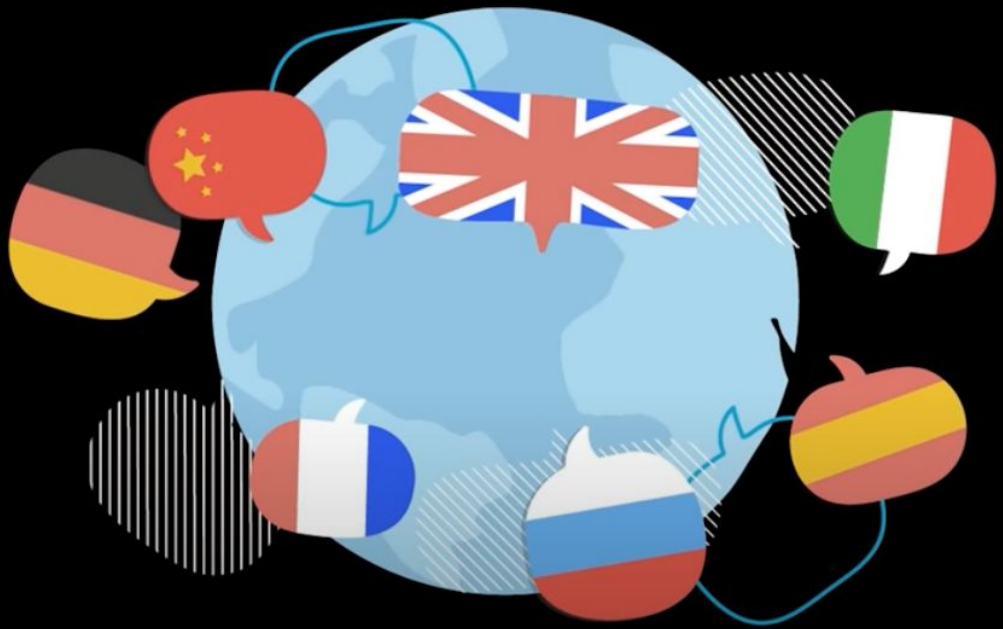
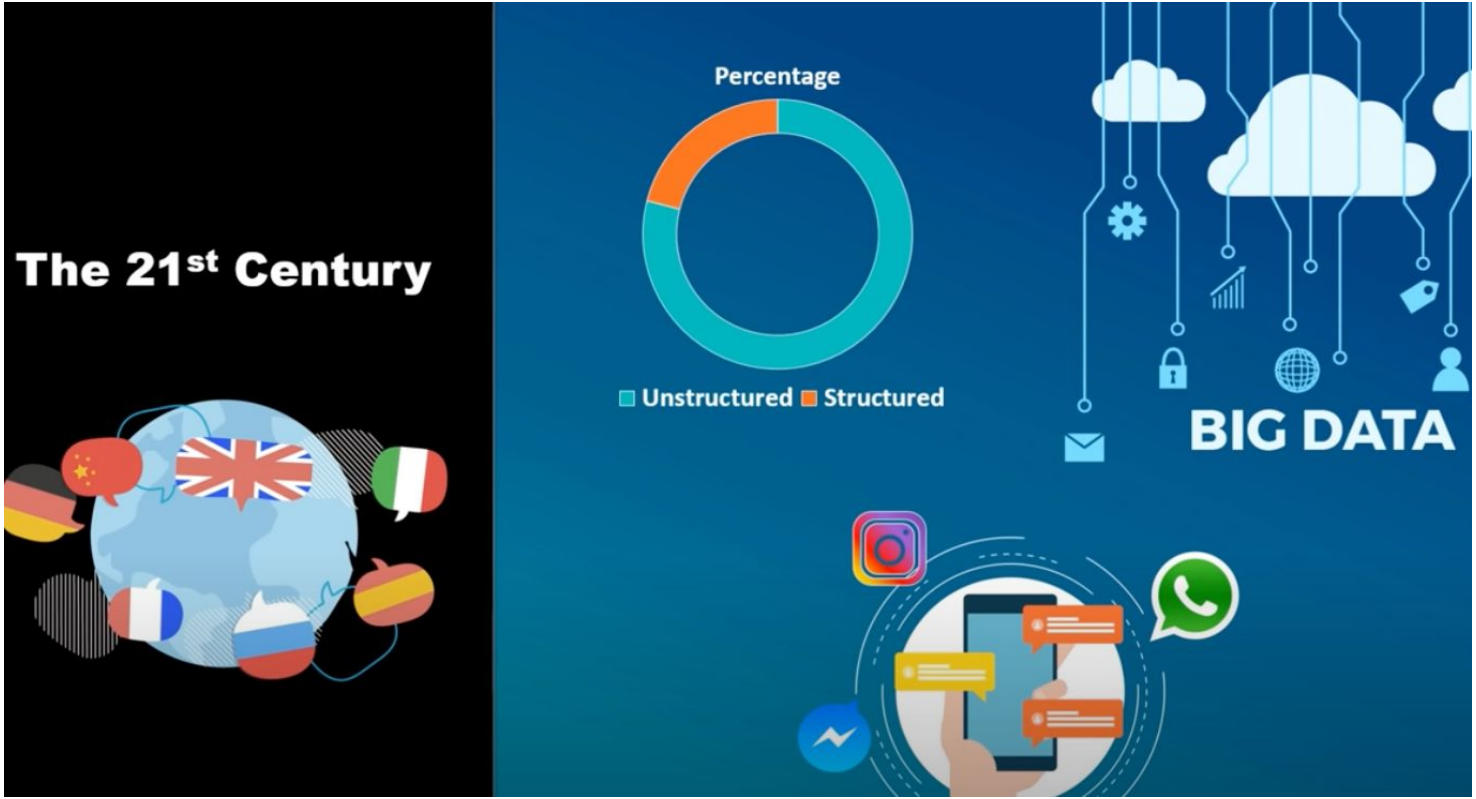
Natural Language Processing

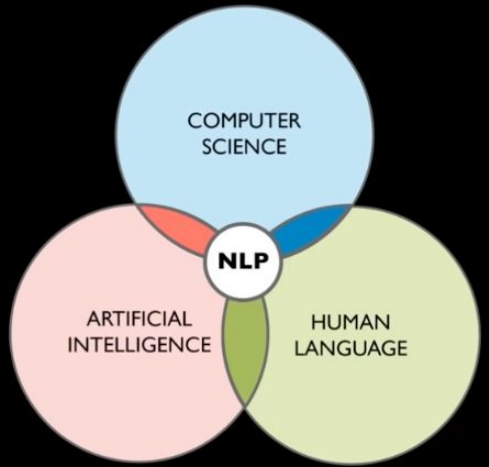
The Human Language




The 21st Century
Percentage
Unstructured
Structured
BIG DATA

Text Mining and NLP

Text Mining / Text Analytics is the process of deriving meaningful information from natural language text



NLP: Natural Language Processing is a part of computer science and artificial intelligence which deals with human languages.

Applications of NLP

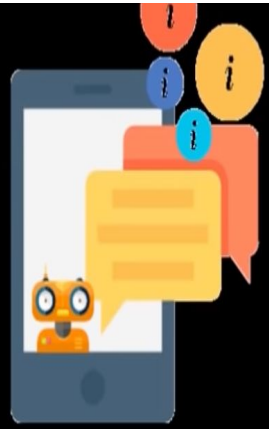
Sentimental Analysis

Applications of NLP

Sentimental Analysis

Chatbot

Sentimental Analysis



Chatbot

Speech Recognition

Machine Translation

Spell Checking

Keyword Searching

Information Extraction

Advertisement Matching

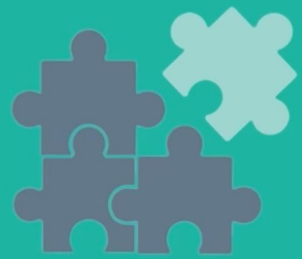
Tokenization

Stemming

Lemmatization

POS Tags

Named Entity Recognition



Chunking

Tokenization

Tokenization | is | the | first | step | in | NLP

Stemming

Normalize words into its base form or root form

Affectation | Affects | Affections | Affected | Affection | Affecting

Stemming

Normalize words into its base form or root form

Affect

Lemmatization

Groups together different inflected forms of a word, called Lemma

Somehow similar to Stemming, as it maps several words into one common root

Output of Lemmatisation is a proper word

For example, a Lemmatiser should map ***gone***, ***going*** and ***went*** into ***go***


POS
infinitives
grammar
english
gerunds
conditionals
negative
questions
adverbs
interrogatives
tenses
worksheets
prepositions
past
nouns
articles
present
exercises
genitive
affirmative
adjectives
relatives
passives
reported
verbs
future
DT
NN
VBD
DT
NN
SENTENCE
The
Dog
killed
the
bat

Named Entity Recognition

MOVIE

MONETARY VALUE

ORGANIZATIO

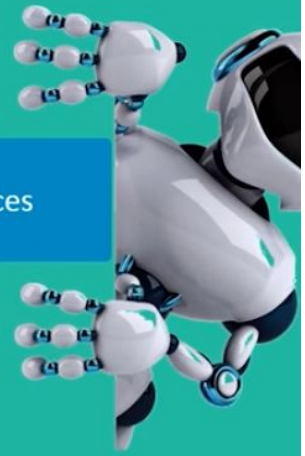
LOCATION

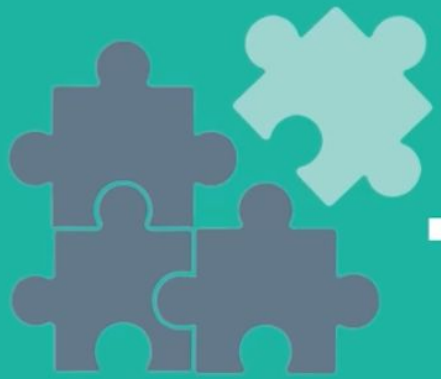
QUANTITIES

PERSON

Chunking

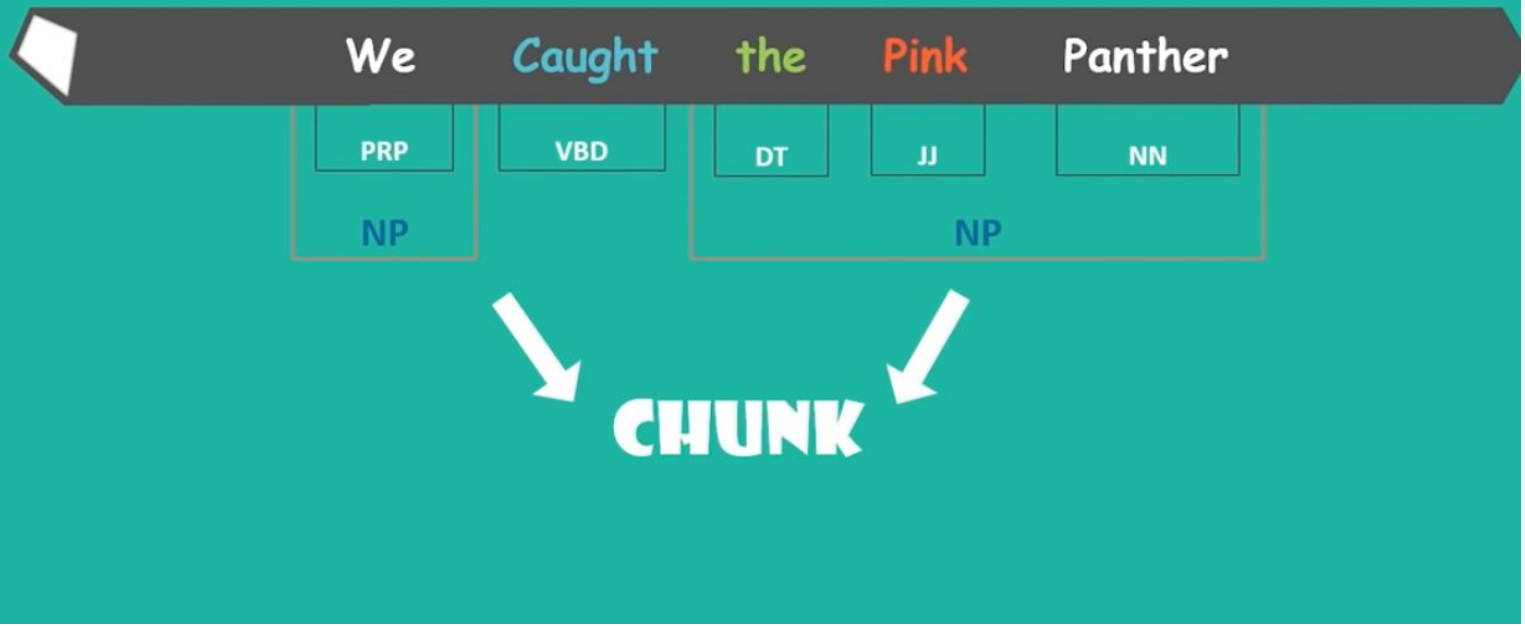
Picking up ***Individual*** pieces of Information and ***Grouping*** them into bigger Pieces





CHUNK

Chunking



Components of NLP

There are the following two components of NLP -

1. Natural Language Understanding (NLU)

Natural Language Understanding (NLU) helps the machine to understand and analyse human language by extracting the metadata from content such as concepts, entities, keywords, emotion, relations, and semantic roles.

NLU mainly used in Business applications to understand the customer's problem in both spoken and written language

NLU involves the following tasks -

- It is used to map the given input into useful representation.
- It is used to analyze different aspects of the language.

Natural Language Generation (NLG)

Natural Language Generation (NLG) acts as a translator that converts the computerized data into natural language representation. It mainly involves Text planning, Sentence planning, and Text Realization.

Note: The NLU is difficult than NLG.

Difference between NLU and NLG

NLU

NLU is the process of reading and interpreting language.
It produces non-linguistic outputs from natural language inputs.

NLG

NLG is the process of writing or generating language.
It produces constructing natural language outputs from non-linguistic inputs.

Sentence Segment is the first step for building the NLP pipeline. It breaks the paragraph into separate sentences.

Example: Consider the following paragraph -

Independence Day is one of the important festivals for every Indian citizen. It is celebrated on the 15th of August each year ever since India got independence from the British rule. The day celebrates independence in the true sense.

Sentence Segment produces the following result:

1. "Independence Day is one of the important festivals for every Indian citizen."
2. "It is celebrated on the 15th of August each year ever since India got independence from the British rule."
3. "This day celebrates independence in the true sense."

Step2: Word Tokenization

Word Tokenizer is used to break the sentence into separate words or tokens.

Example:

COMPANY offers Corporate Training, Summer Training, Online Training, and Winter Training.

Word Tokenizer generates the following result:

“COMPANY", "offers", "Corporate", "Training", "Summer", "Training", "Online", "Training", "and", "Winter", "Training", "."

Step3: Stemming

Stemming is used to normalize words into its base form or root form. For example, celebrates, celebrated and celebrating, all these words are originated with a single root word "celebrate." The big problem with stemming is that sometimes it produces the root word which may not have any meaning.

For Example, intelligence, intelligent, and intelligently, all these words are originated with a single root word "intelligent." In English, the word "intelligent" do not have any meaning.

Step 4: Lemmatization

Lemmatization is quite similar to the Stamming. It is used to group different inflected forms of the word, called Lemma. The main difference between Stemming and lemmatization is that it produces the root word, which has a meaning.

For example: In lemmatization, the words intelligence, intelligent, and intelligently has a root word intelligent, which has a meaning.

Step 5: Identifying Stop Words

In English, there are a lot of words that appear very frequently like "is", "and", "the", and "a". NLP pipelines will flag these words as stop words. **Stop words** might be filtered out before doing any statistical analysis.

Example: He **is a** good boy.

Note: When you are building a rock band search engine, then you do not ignore the word "The."

Step 6: Dependency Parsing

Dependency Parsing is used to find that how all the words in the sentence are related to each other.

Step 7: POS tags

POS stands for parts of speech, which includes Noun, verb, adverb, and Adjective. It indicates that how a word functions with its meaning as well as grammatically within the sentences. A word has one or more parts of speech based on the context in which it is used.

Example: "Google" something on the Internet.

In the above example, Google is used as a verb, although it is a proper noun.

Step 8: Named Entity Recognition (NER)

Named Entity Recognition (NER) is the process of detecting the named entity such as person name, movie name, organization name, or location.

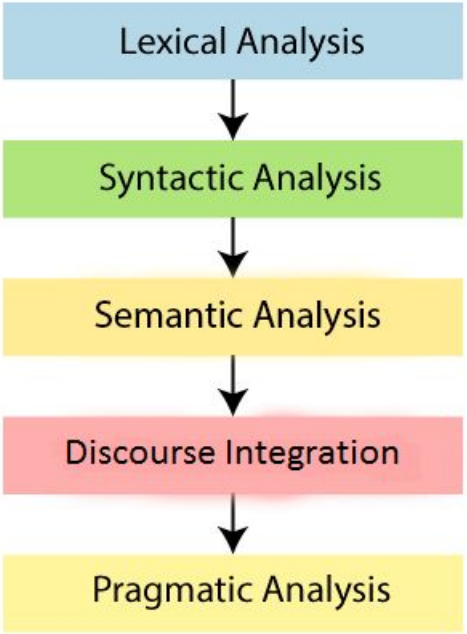
Example: Steve Jobs introduced iPhone at the Macworld Conference in San Francisco, California.

Step 9: Chunking

Chunking is used to collect the individual piece of information and grouping them into bigger pieces of sentences.

Phases of NLP

There are the following five phases of NLP:



1. Lexical Analysis and Morphological

The first phase of NLP is the Lexical Analysis. This phase scans the source code as a stream of characters and converts it into meaningful lexemes. It divides the whole text into paragraphs, sentences, and words.

2. Syntactic Analysis (Parsing)

Syntactic Analysis is used to check grammar, word arrangements, and shows the relationship among the words.

Example: Agra goes to the Poonam
In the real world, Agra goes to the Poonam, does not make any sense, so this sentence is rejected by the Syntactic analyzer.

3. Semantic Analysis

Semantic analysis is concerned with the meaning representation. It mainly focuses on the literal meaning of words, phrases, and sentences.

4. Discourse Integration

Discourse Integration depends upon the sentences that proceeds it and also invokes the meaning of the sentences that follow it.

5. Pragmatic Analysis

Pragmatic is the fifth and last phase of NLP. It helps you to discover the intended effect by applying a set of rules that characterize cooperative dialogues.

For Example: "Open the door" is interpreted as a request instead of an order.

Why NLP is difficult?

NLP is difficult because Ambiguity and Uncertainty exist in the language.

Ambiguity

There are the following three ambiguity -

- **Lexical Ambiguity**

Lexical Ambiguity exists in the presence of two or more possible meanings of the sentence within a single word.

Example:

Manya is looking for a **match**.

In the above example, the word match refers to that either Manya is looking for a partner or Manya is looking for a match. (Cricket or other match)

- **Syntactic Ambiguity**

Syntactic Ambiguity exists in the presence of two or more possible meanings within the sentence.

Example:

I saw the girl with the binocular.

In the above example, did I have the binoculars? Or did the girl have the binoculars?

- **Referential Ambiguity**

Referential Ambiguity exists when you are referring to something using the pronoun.

Example: Kiran went to Sunita. She said, "I am hungry."

In the above sentence, you do not know that who is hungry, either Kiran or Sunita.

•Some interpretations of : I made her duck.

1. I cooked duck for her.
2. I cooked duck belonging to her.
3. I created a toy duck which she owns.
4. I caused her to quickly lower her head or body.
5. I used magic and turned her into a duck.

Regular Expressions

Regular expression search requires a **pattern** that we want to search for, and a **cor- pus** of texts to search through.

A regular expression search function will search through the corpus returning all texts that contain the pattern.

The simplest kind of regular expression is a sequence of simple characters. For ex- ample, to search for *woodchuck*, we type /woodchuck/.

So the regular expression /Buttercup/ matches any string containing the substring *Buttercup*

RE	Example Patterns Matched
/woodchucks/	"interesting links to woodchucks and lemurs"
/a/	"Mary Ann stopped by Mona's"
/Claire_says,/	" "Dagmar, my gift please," Claire says,"
/DOROTHY/	"SURRENDER DOROTHY"
/!/	"You've left the burglar behind again!" said Nori

Regular expressions are case sensitive; lowercase /s/ is distinct from uppercase /S/ (/s/ matches a lower case *s* but not an uppercase *S*).

This means that the pattern /woodchucks/ will not match the string *Woodchucks*. We can solve this problem with the use of the square braces [and].

The string of characters inside the braces specify a disjunction of characters to match. For example next table shows that the pattern /[wW]/ matches patterns containing either *w* or *W*.

RE	Match	Example Patterns Matched
`/[A-Z]/`	an uppercase letter	"we should call it 'Drenched Blossoms'"
`/[a-z]/`	a lowercase letter	"my beans were impatient to be hoed!"
`/[0-9]/`	a single digit	"Chapter 1: Down the Rabbit Hole"

the brackets can be used with the dash (-) to specify any one character in a **range**.

RE	Match (single characters)	Example Patterns Matched
`[^A-Z]`	not an uppercase letter	"Oyfn pripetchik"
`[^Ss]`	neither 'S' nor 's'	"I have no exquisite reason for't"
`[^\.]`	not a period	"our resident Djinn"
`[e^]`	either 'e' or '^'	"look up ^ now"
`a^b`	the pattern 'a^b'	"look up a^ b now"

RE	Match	Example Patterns Matched
`woodchucks?`	woodchuck or woodchucks	"woodchuck"
`colou?r`	color or colour	"colour"

RE	Match	Example Patterns
`/beg.n/`	any character between *beg* and *n*	begin, beg'n, begun

Kleene +, which means "one or more of the previous character". Thus the expression /[0-9]+/ is the normal way to specify "a sequence of digits". There are thus two ways to specify the sheep language: /baaa*!/ or /baa+!/.

One very important special character is the period (/./), a **wildcard** expression that matches any single character (*except* a carriage return)

For example suppose we want to find any line in which a particular word, for example *aardvark*, appears twice. We can specify this with the regular expression /aardvark.*aardvark/.

Anchor are special characters that anchor regular expressions to particular places in a string. The most common anchors are the caret ˆ and the dollar-sign $.

The caret ˆ matches the start of a line. The pattern /ˆThe/ matches the word *The* only at the start of a line. Thus there are three uses of the caret ˆ: to match the start of a line, as a negation inside of square brackets, and just to mean a caret. (What are the contexts that allow Perl to know which function a given caret is supposed to have?)

The dollar sign $ matches the end of a line. So the pattern $ is a useful pattern for matching a space at the end of a line, and /ˆThe dog\.$/ matches a line that contains only the phrase *The dog.*

(We have to use the backslash here since we want the . to mean “period” and not the wildcard.)

There are also two other anchors: \b matches a word boundary, while \B matches a non-boundary. Thus /\bthe\b/ matches the word *the* but not the word *other*.

Morphology

Morphology is the study of the way words are built up from smaller meaning- bearing units, **morphemes.**

o for example the word *fox* consists of a single morpheme (the morpheme *fox*) while the word *cats* consists of two: the morpheme *cat* and the morpheme *-s*.

two broad classes of morphemes: **stems** and **affixes.**

the stem is the ‘main’ morpheme of the word, supplying the main meaning, while the affixes add ‘additional’ meanings of various kinds.

- Eg. Mangoes -> Mango and es

Morphology

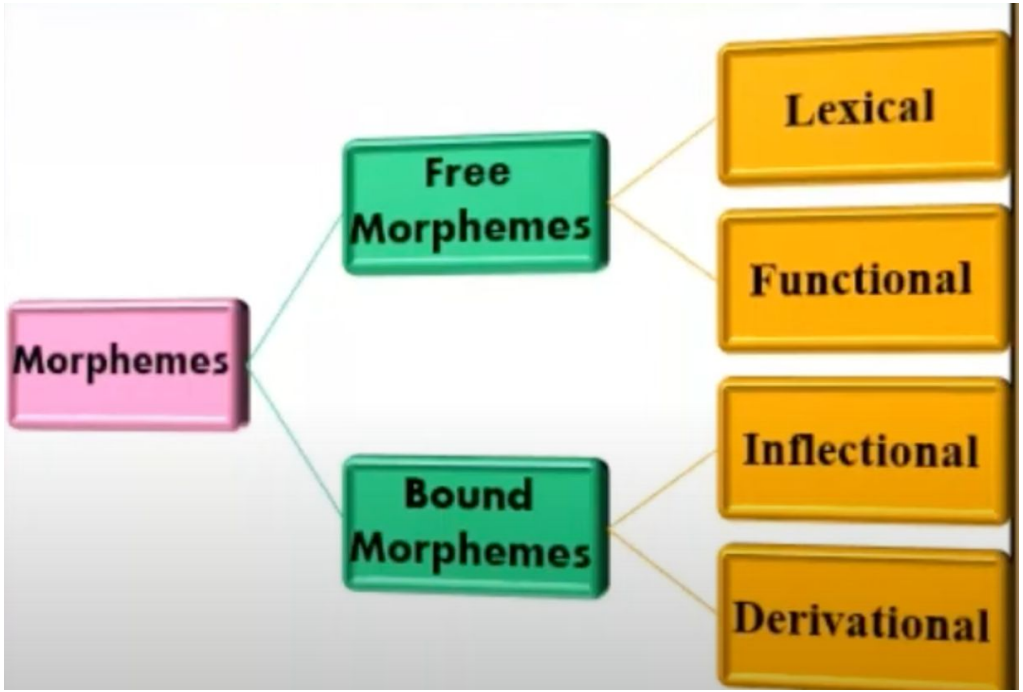
- Morphology is the study of the way words are built from smaller meaningful units called morphemes.

- We can divide morphemes into two broad classes.
  - Stems–thecoremeaningfulunits,therootoftheword.
  - Affixes–addadditionalmeaningsandgrammaticalfunctionstowords.

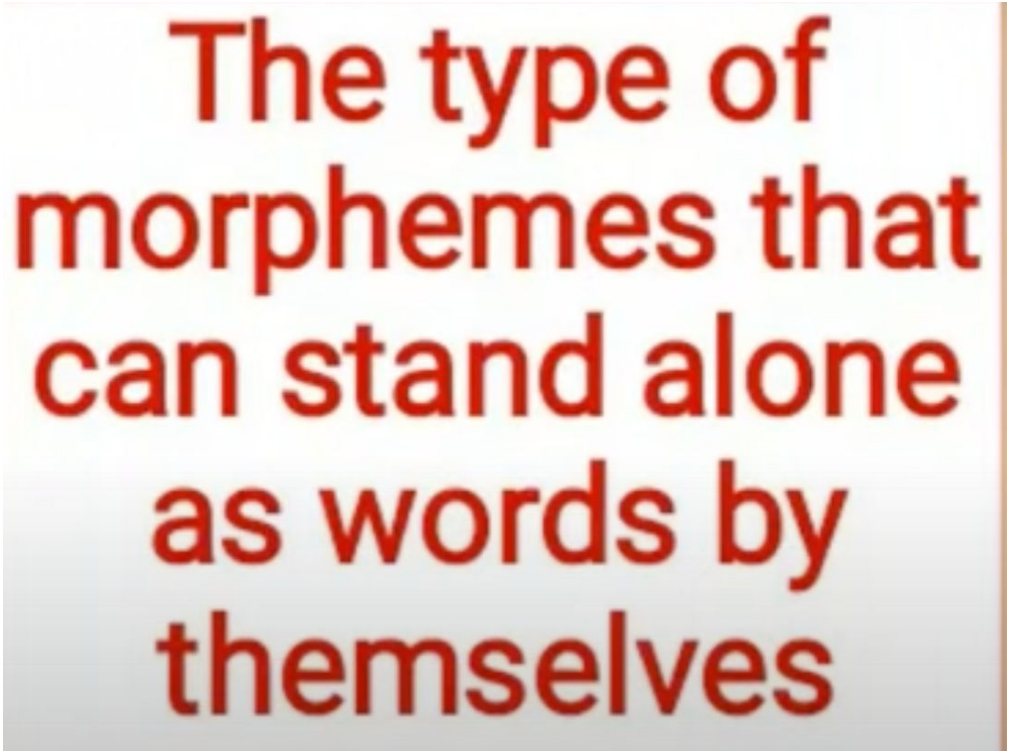
- Affixes are further divided into:
  - Prefixes–precedethestem: do/undo, reformed

- – Suffixes–followthestem: eat/eats, loved
  - Infixes–areinsertedinsidethestem
  - Circumfixes–precede and followthestem

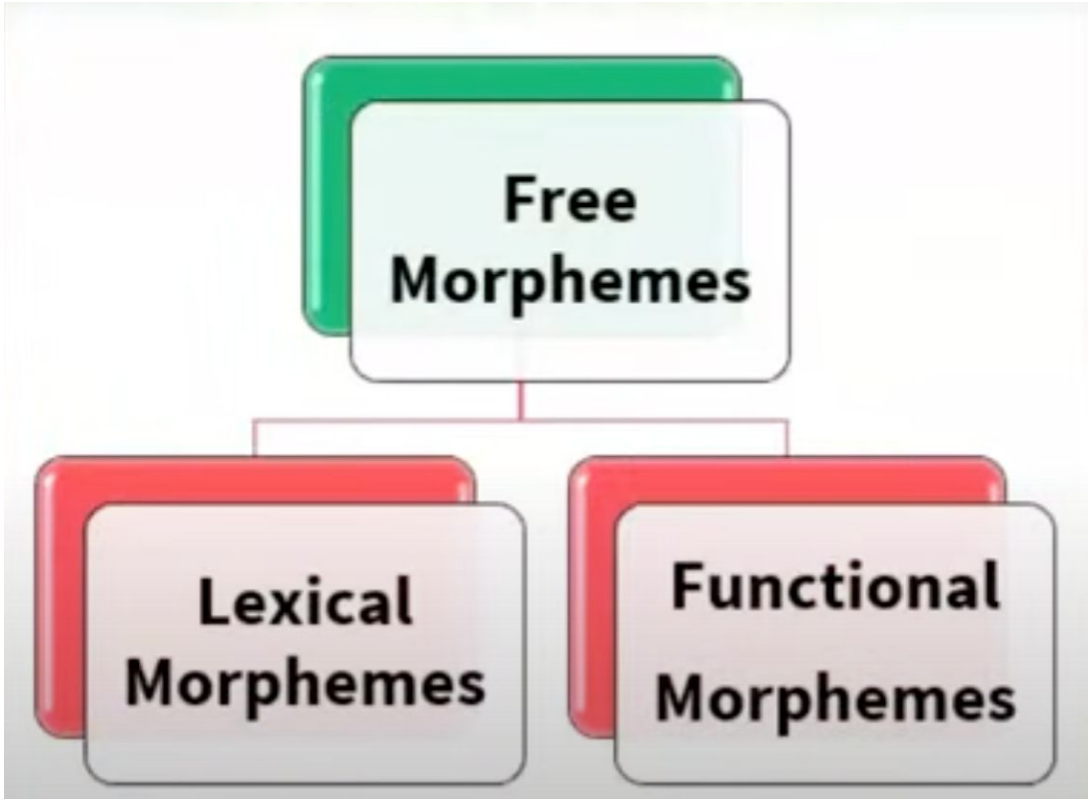
Types


Morphemes
Free Morphemes
Bound Morphemes
Lexical
Functional
Inflectional
Derivational

Free Morphene



Examples of Free Morphemes	
Boy	
Girl	
Car	
Beauty	


Free Morphemes
Lexical Morphemes
Functional Morphemes

Lexical Morphene

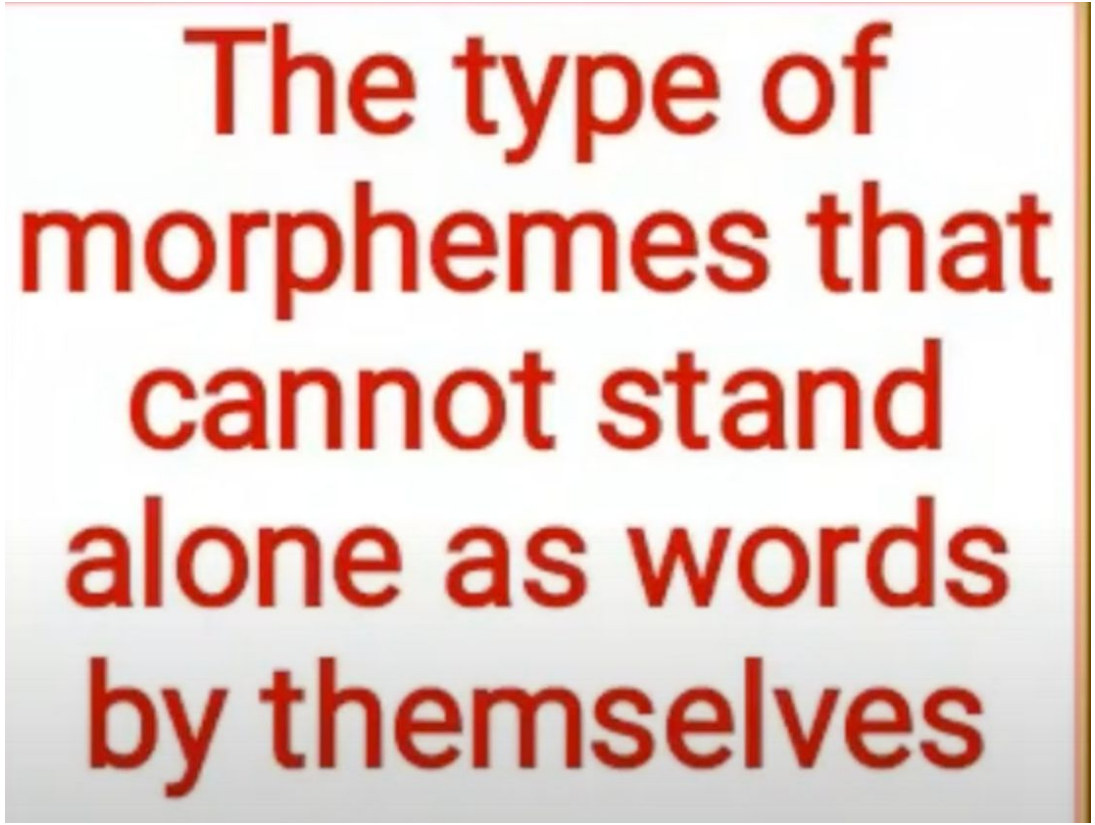
Examples of Lexical Free Morphemes	
Cup	Loyal
Glass	Fast
Home	Faith
Table	Work
Drink	Sleep

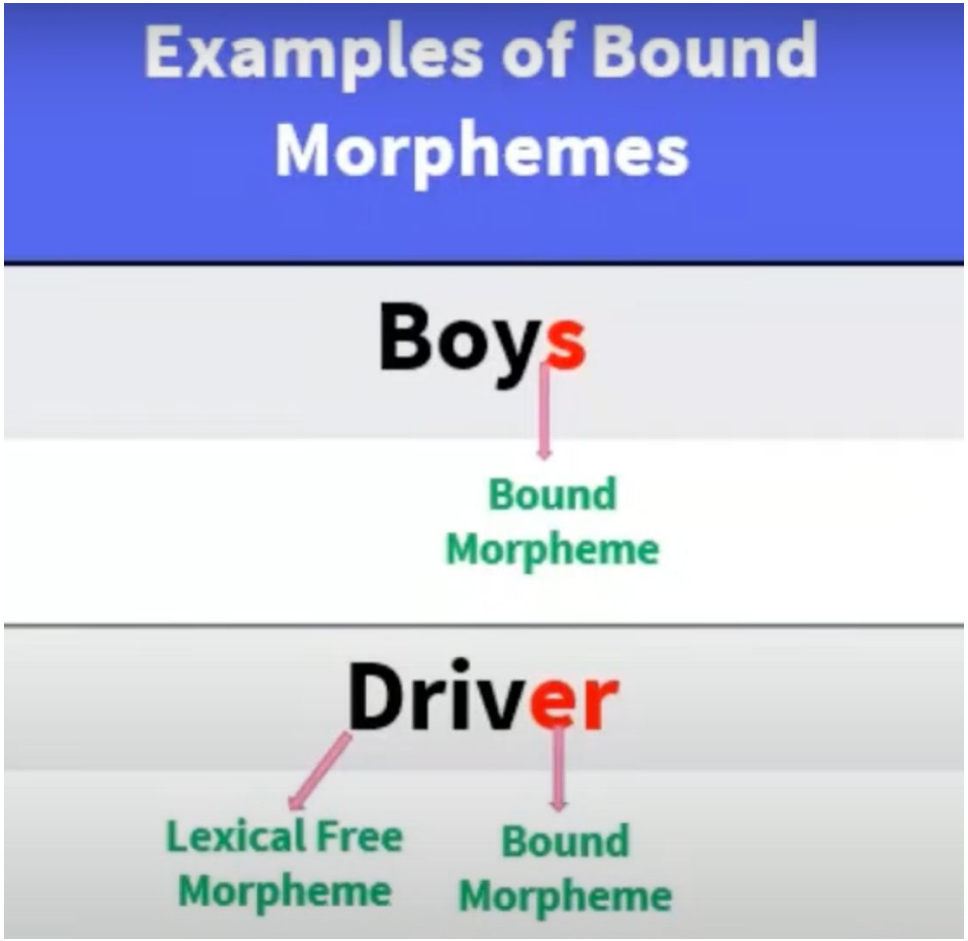
Functional Morphene

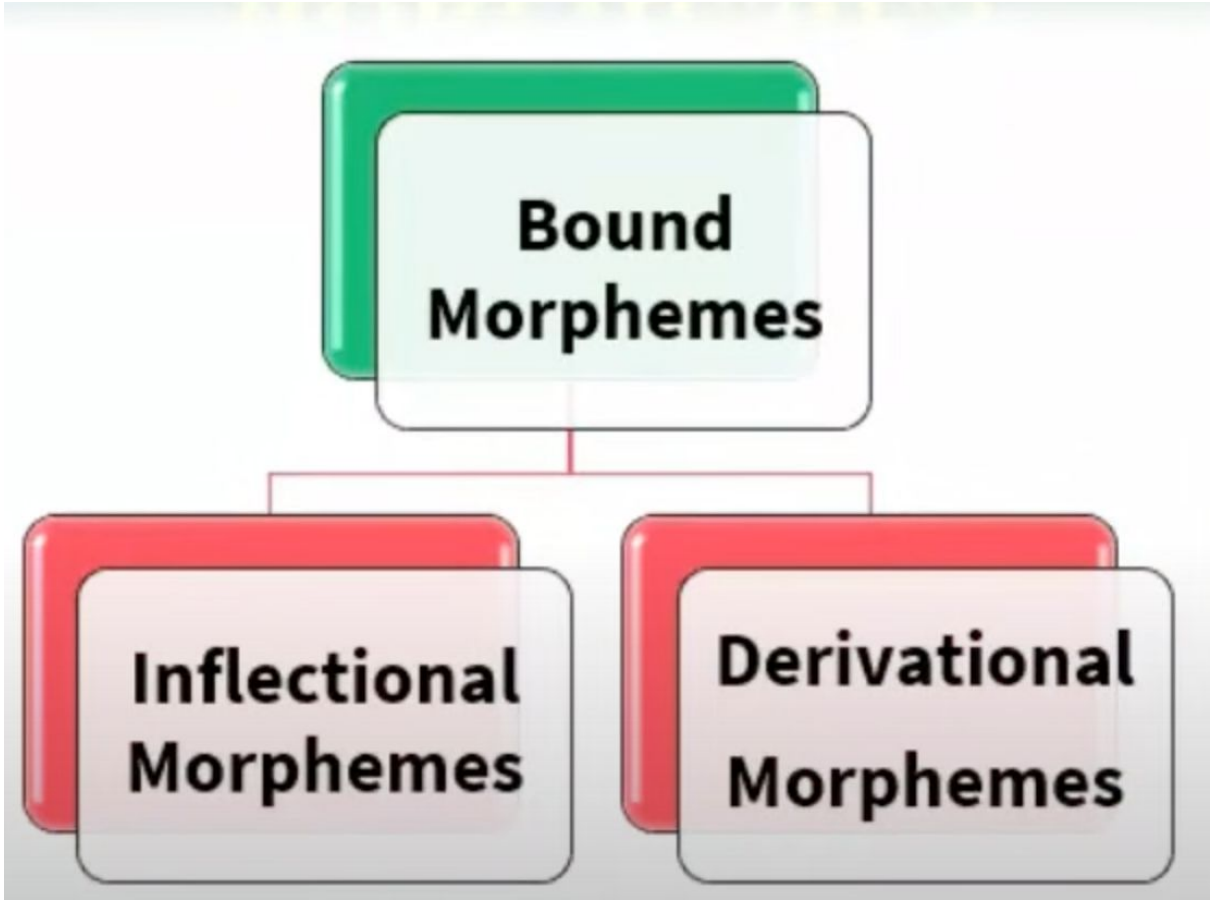
Examples of Functional Free Morphemes

Prepositions	Conjunctions
Ex: Of, To	Ex: And, But
Articles	Determiners
Ex: A, An, The	Ex: This, That
Pronouns	Auxiliary Verbs
Ex: I, You	Ex: Is, Can

Bound Morphene

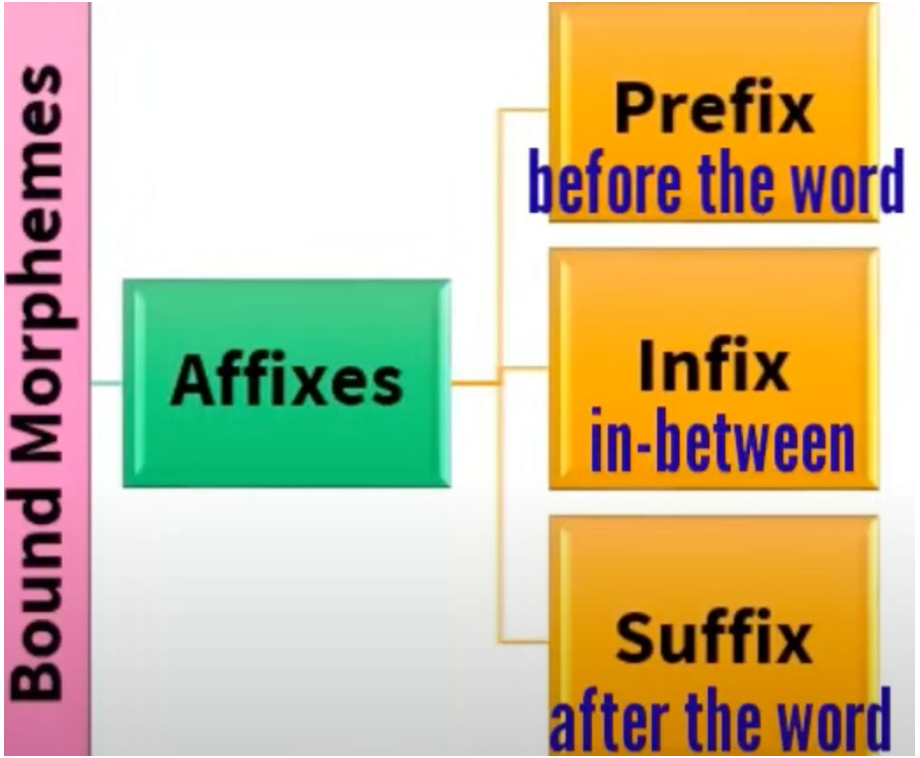



Examples of Bound Morphemes
Boys
Bound Morpheme
Driver
Lexical Free Morpheme
Bound Morpheme


Bound Morphemes
Inflectional Morphemes
Derivational Morphemes

Bound Morphene

These are AFFIXES that must be attached to the word.



- Free Morphemes: lexemes and grammatical morphemes.
- Eg Wash and ing
- Bound Morphemes: Inflectional and Derivational Morphology
- Eg: Car and Cars (N)
- Eg Danger (N) and Dangerous (Adj)
- Class Maintaining: No change in POS tag
- Eg. Law (N) and Lawyer(N)

Inflectional and Derivational Morphology

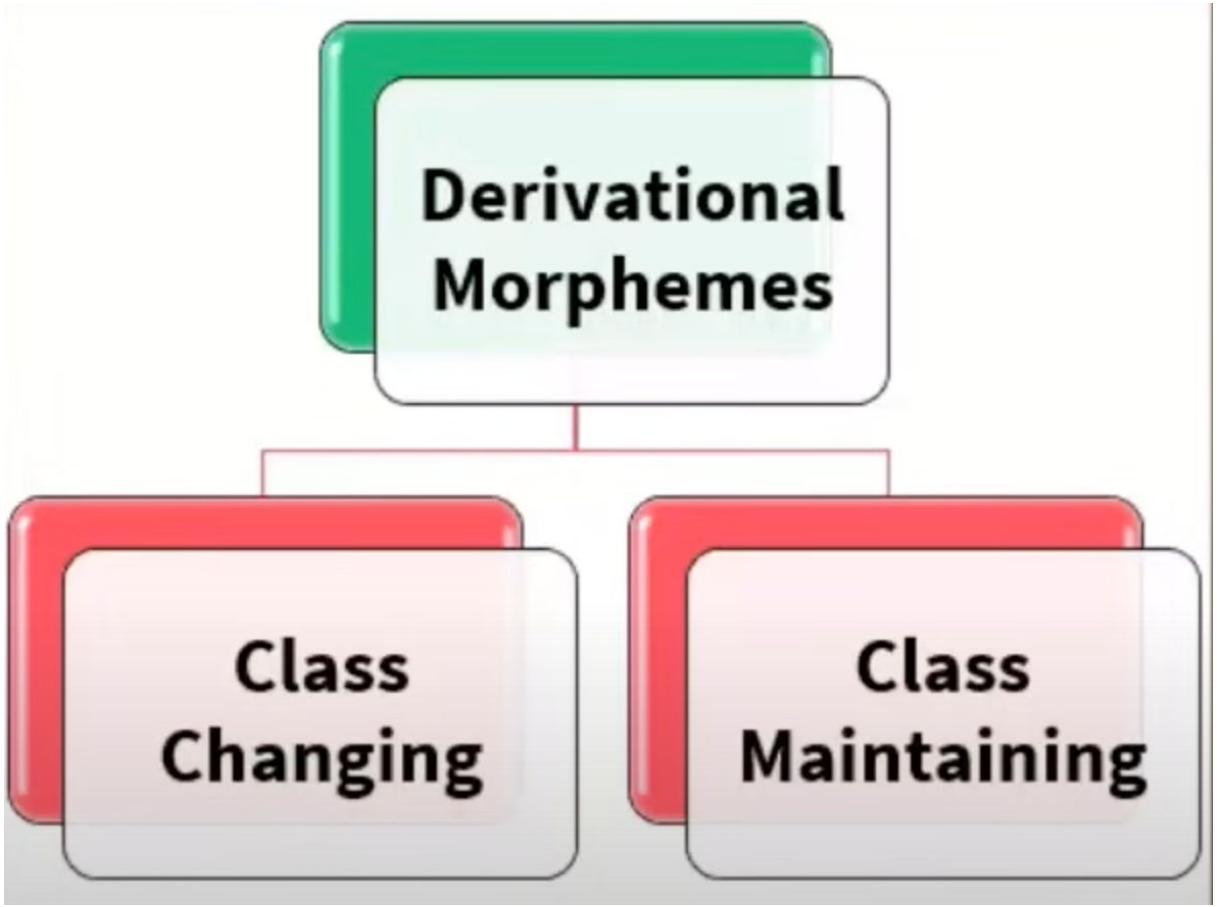
- There are two broad classes of morphology: –
  Inflectionalmorphology
  – Derivationalmorphology

- After a combination with an inflectional morpheme, the meaning and class of the actual stem usually do not change.
  – eat/eats pencil/pencils

- After a combination with an derivational morpheme, the meaning and the class of the actual stem usually change.

do / undo friend / friendly

- The irregular changes may happen with derivational affixes.


Derivational Morphemes
Class Changing
Class Maintaining

Examples of Derivational Morphemes	
Drive	Driver
Verb	Noun
Beauty	Beautiful
Noun	Adjective
Valid	Invalid
Noun	Noun

Examples of Class Maintaining Derivational Morphemes

Valuable	Invaluable
Adjective	Adjective
Valid	Invalid
Noun	Noun

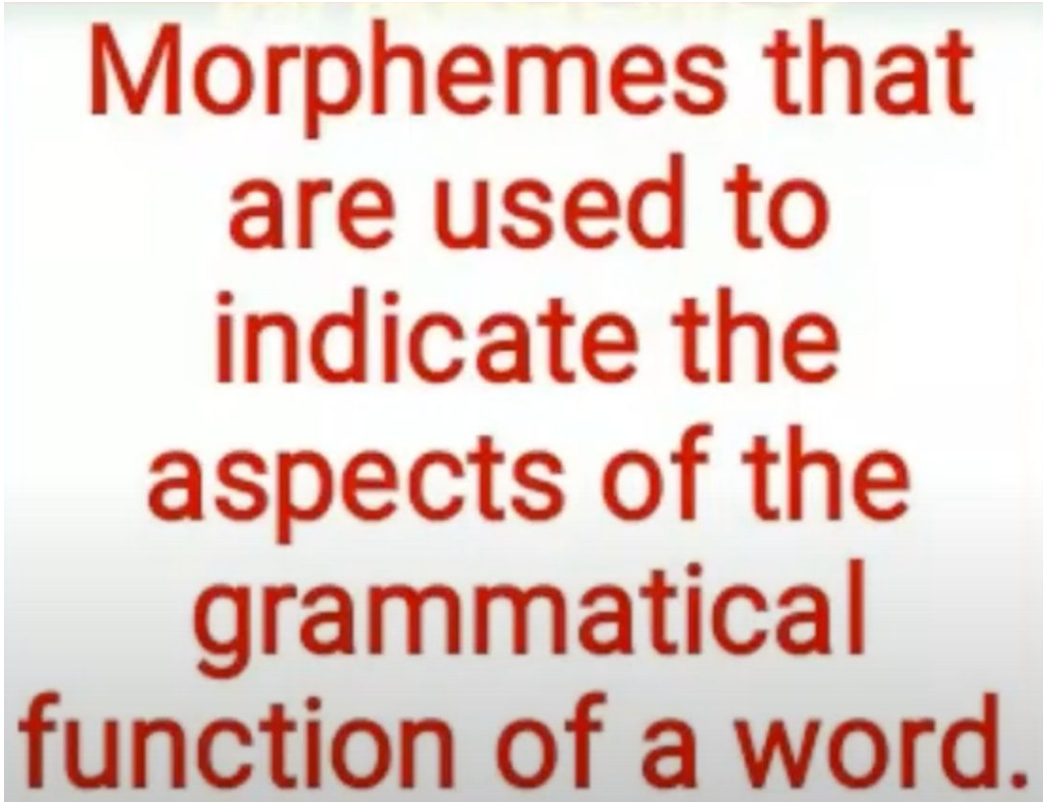
Examples of Class Changing Derivational Morphemes

Help	Helper
Verb	Noun
Help	Helpful
Verb	Adjective
Adjust	Adjustment
Verb	Noun

English Derivational Morphology

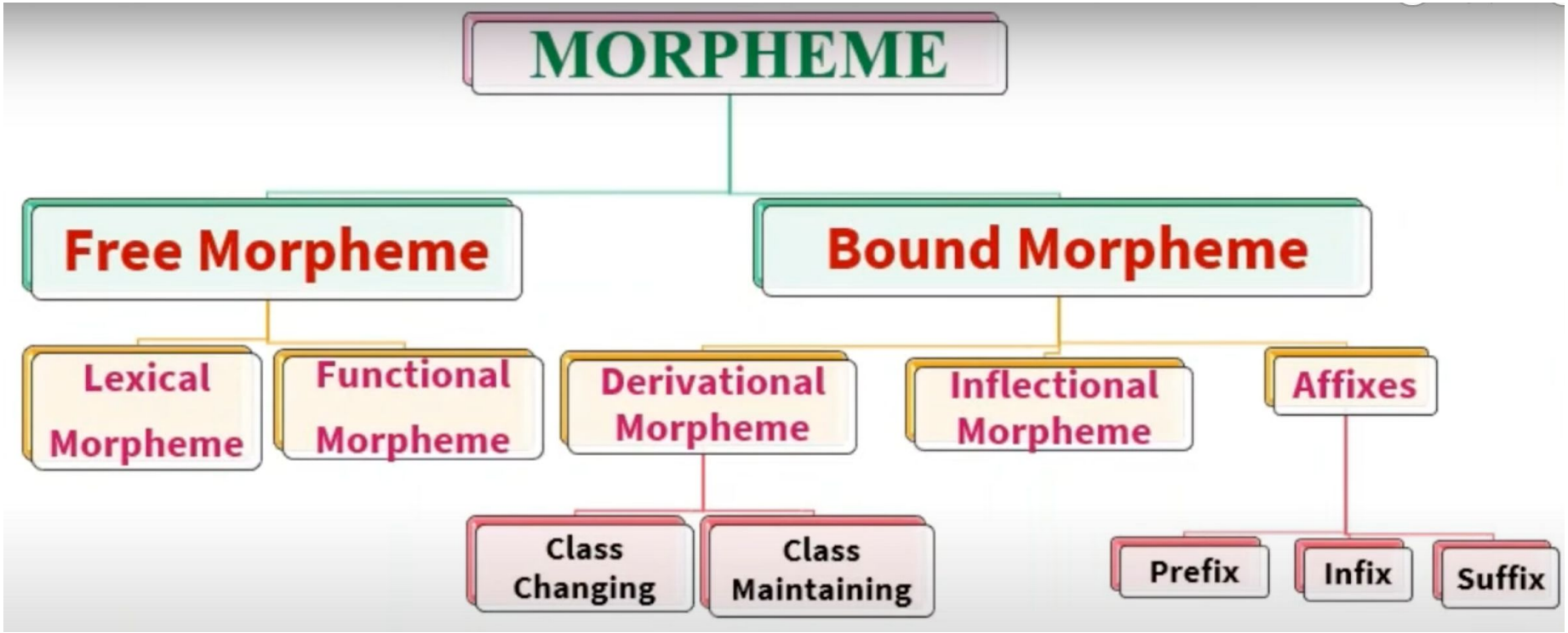
- Some English derivational affixes
  - – -ation : transport / transportation
  - – -er:kill/killer
  - – -ness : fuzzy / fuzziness
  - – -al:computation/computational
  - – -able:break/breakable
  - – -less : help / helpless
  - – un : do / undo

Inflectional Morphenes



INFLECTIONALS

Noun	Plural form (s)	Boys
	Possessive noun (‘s)	Boy’s book
Verb	3rd person singular (s)	He helps me
	Progressive verb (-ing)	Coming
	Past tense (-ed)	Reached
	Past Participle (-en)	Eaten
Adjective	Comparative (-er)	Happier
	Superlative (-est)	Happiest


MORPHEME
Free Morpheme
Bound Morpheme
Lexical Morpheme
Functional Morpheme
Derivational Morpheme
Inflectional Morpheme
Affixes
Class Changing
Class Maintaining
Prefix
Infix
Suffix

Finite-State Automata

a regular expression is one way of describing a finite-state automaton (FSA).

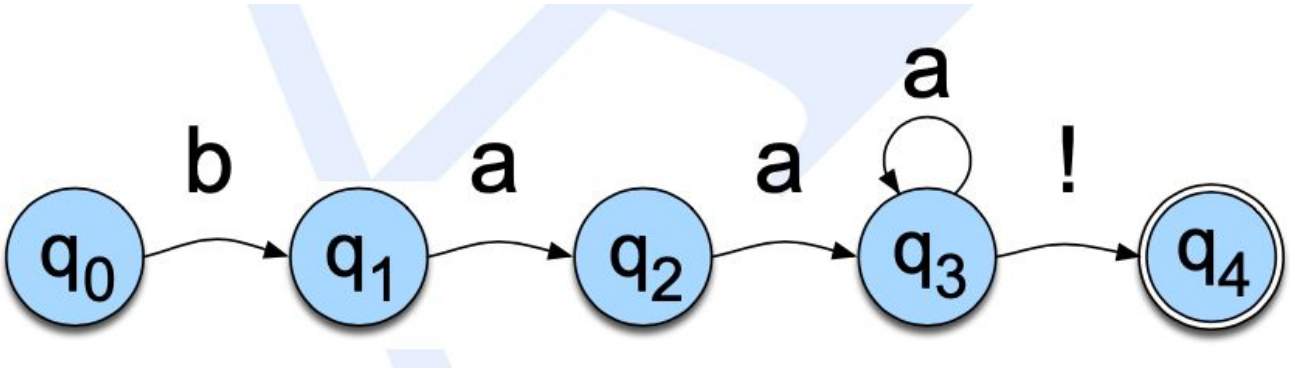
Both regular expressions and finite- state automata can be used to describe regular languages.

Using an FSA to Recognize Sheeptalk

Let's begin with the "sheep language"

we defined the sheep language as any string from the following (infinite) set:

baa! baaa! baaaa! baaaaa! ...



The regular expression for this kind of “sheeptalk” is /baa+!/.

Fig shows an **automaton** for modeling this regular expression. The automaton (i.e., machine, also called **finite automaton**, **finite-state automaton**, or **FSA**) recognizes a set of strings, in this case the strings characterizing sheep talk, in the same way that a regular expression does.

We represent the automaton as a directed graph:

A finite set of vertices (also called nodes), together with a set of directed links between pairs of vertices called arcs. We’ll represent vertices with circles and arcs with arrows.

The automaton has five **states**, which are represented by nodes in the graph. State 0 is the **start state**. In our examples state 0 will generally be the start state; to mark another state as the start state we can add an incoming arrow to the start state. State 4 is the **final state** or **accepting state**, which we represent by the double circle. It also has four **transitions**, which we represent by arcs in the graph.

Morphological parsing

Morphological parsing, in natural language processing, is the process of determining the morphemes from which a given word is constructed. It must be able to distinguish between orthographic rules and morphological rules. For example, the word 'foxes' can be decomposed into 'fox' (the stem), and 'es' (a suffix indicating plurality).

The generally accepted approach to morphological parsing is through the use of a finite state transducer (FST), which inputs words and outputs their stem and modifiers. The FST is initially created through algorithmic parsing of some word source, such as a dictionary, complete with modifier markups.

With the advancement of neural networks in natural language processing, it became less common to use FST for morphological analysis, especially for languages for which there is a lot of available training data. For such languages, it is possible to build character-level language models without explicit use of a morphological parse

Orthographic

Orthographic rules are general rules used when breaking a word into its stem and modifiers.

An example would be: singular English words ending with -y, when pluralized, end with -ies. Contrast this to morphological rules which contain corner cases to these general rules.

Both of these types of rules are used to construct systems that can do morphological parsing.

Morphological

Morphological rules are exceptions to the orthographic rules used when breaking a word into its stem and modifiers. An example would be while one normally pluralizes a word in English by adding 's' as a suffix, the word 'fish' does not change when pluralized.

Contrast this to orthographic rules which contain general rules. Both of these types of rules are used to construct systems that can do morphological parsing.

Various models of natural morphological processing have been proposed. Some experimental studies suggest that monolingual speakers process words as wholes upon listening to them, while their late bilinguals peers break words down into their corresponding morphemes, because their lexical representations are not as specific, and because lexical processing in the second language may be less frequent than processing the mother tongue. [2]

Applications of morphological processing include machine translation, spell checker, and information retrieval.

Introduction

- Consider a simple example: parsing just the productive nominal plural (*-s*) and the verbal progressive (*-ing*).
- Our goal will be to take input forms like those in the first column below and produce output forms like those in the second column.
- The second column contains the stem of each word as well as assorted morphological **features**.
- These features specify additional information about the stem.
  - E.g. +SG → singular, +PL → plural

Input	Morphological Parsed Output
cats	cat +N +PL
cat	cat +N +SG
cities	city +N +PL
geese	goose +N +PL
goose	(goose +N +SG) or (goose +V)
gooses	goose +V +3SG
merging	merge +V +PRES-PART
caught	(catch +V +PAST-PART) or (catch +V +PAST)

Introduction (Cont…)

In order to build a morphological parser, we'll need at least the following:

- **A lexicon:**
  - The list of stems and affixes, together with basic information about them (whether a stem is a Noun stem or a Verb stem, etc).
- **Morphotactics:**
  - The model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word.
  - For example, the rule that the English plural morpheme follows the noun rather than preceding it.
- **Orthographic rules:**
  - These **spelling rules** are used to model the changes that occur in a word, usually when two morphemes combine (for example the y →ie spelling rule that changes *city + -s* to cities rather than citys).

The Lexicon and Morphotactics

A lexicon is a repository for words.

The simplest possible lexicon would consist of an explicit list of every word of the language (*every* word, i.e. including abbreviations ('AAA') and proper names ('Jane' or 'Beijing') as follows:

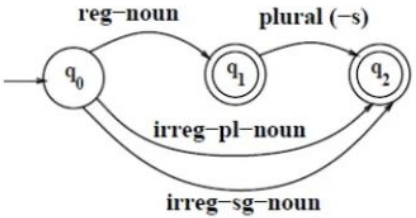
a
AAA
AA
Aachen
aardvark
aardwolf
aba
abaca
aback
. . .

The Lexicon and Morphotactics (Cont…)

- Since it will often be inconvenient or impossible, for the various reasons , to list every word in the language, computational lexicons are usually structured with a list of each of the stems and affixes of the language together with a representation of the morphotactics that tells us how they can fit together.
- There are many ways to model morphotactics; one of the most common is the finite-state automaton.

The Lexicon and Morphotactics (Cont…)

- The following FSA assumes that the lexicon includes regular nouns (**reg-noun**) that take the regular *-s* plural (e.g. *cat*, *dog*, *fox*, *aardvark*), also includes irregular noun forms that don't take *-s*, both singular **irreg-sg-noun** (*goose, mouse*) and plural **irreg-pl-noun** (*geese, mice*).

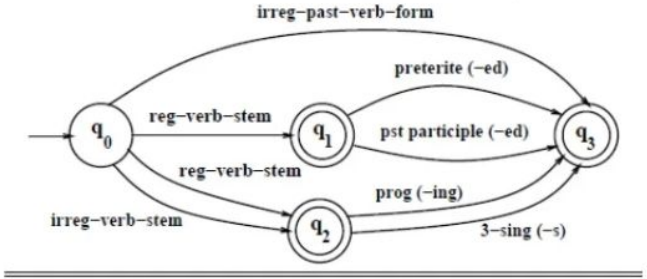


A finite-state automaton for English nominal inflection.

reg-noun	irreg-pl-noun	irreg-sg-noun	plural
fox	geese	goose	-s
cat	sheep	sheep	
dog	mice	mouse	
aardvark			

The Lexicon and Morphotactics (Cont…)

- A similar model for English verbal inflection is



A finite-state automaton for English verbal inflection

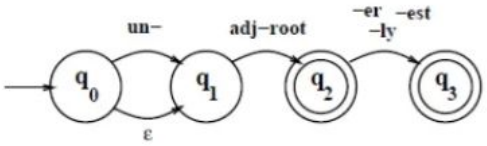
This lexicon has three stem classes (reg-verb-stem, irreg-verb-stem, and irreg-past-verb-form), plus 4 more affix classes (*-ed* past, *-ed* participle, *-ing* participle, and 3rd singular *-s*)

reg-verb-stem	irreg-verb-stem	irreg-past-verb	past	past-part	pres-part	3sg
walk	cut	caught	-ed	-ed	-ing	-s
fry	speak	ate				
talk	sing	eaten				
impeach	sang					
	cut					
	spoken					

The Lexicon and Morphotactics (Cont…)

- Small part of the morphotactics of English adjectives;

big, bigger, biggest
cool, cooler, coolest, coolly
red, redder, reddest
clear, clearer, clearest, clearly, unclear, unclearly
happy, happier, happiest, happily
unhappy, unhappier, unhappiest, unhappily
real, unreal, really

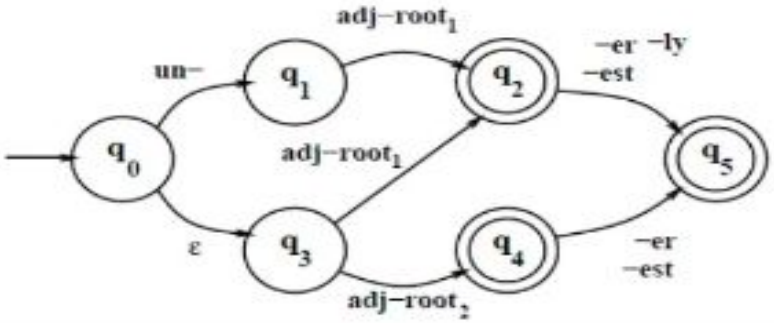


An FSA for a fragment of English adjective morphology: Antworth's Proposal #1.

- An initial hypothesis might be that adjectives can have an optional prefix (*un-*), an obligatory root (*big, cool*, etc) and an optional suffix (*-er, -est*, or *-ly*).

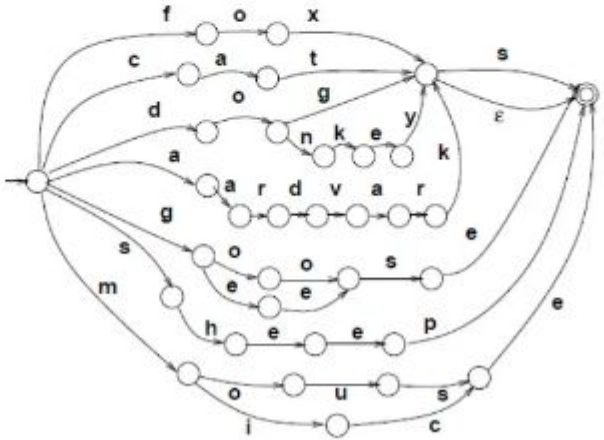
The Lexicon and Morphotactics (Cont...)

- While previous FSA will recognize all the adjectives in the given table, it will also recognize ungrammatical forms like *unbig*, *redly*, and *realest*.
- We need to set up classes of roots and specify which can occur with which suffixes.
- So **adj-root**1 would include adjectives that can occur with *un-* and *-ly* (*clear*, *happy*, and *real*) while **adj-root**2 will include adjectives that can't (*big*, *cool*, and *red*).



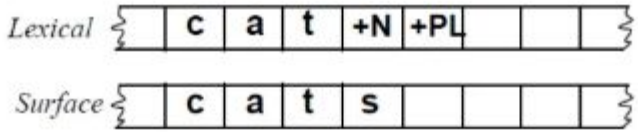
The Lexicon and Morphotactics (Cont…)

- Noun Recognition FSA



Morphological Parsing with Finite-State Transducers

- Given the input *cats*, we'd like to output cat +N +PL, telling us that cat is a plural noun.
- We will do this via a version of **two-level morphology**, first proposed by Koskenniemi (1983).
- Two level morphology represents a word as a correspondence between a **lexical level**, which represents a simple concatenation of morphemes making up a word, and the **surface** level, which represents the actual spelling of the final word.
- Morphological parsing is implemented by building mapping rules that map letter sequences like *cats* on the surface level into morpheme and features sequences like cat +N +PL on the lexical level.



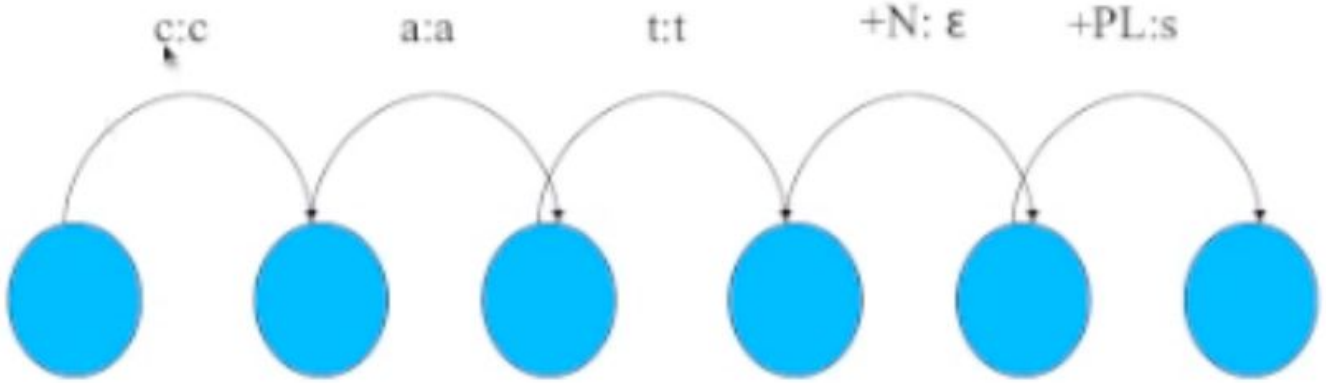
Morphological Parsing with Finite-State Transducers (Cont…)

- The automaton that we use for performing the mapping between lexical and surface levels is the **finite-state transducer** or **FST**.
- A transducer maps between FST one set of symbols and another; a finite-state transducer does this via a finite automaton.
- Thus we usually visualize an FST as a two-tape automaton which recognizes or generates *pairs* of strings.
- The FST thus has a more general function than an FSA; where an FSA defines a formal language by defining a set of strings, an FST defines a *relation* between sets of strings.
- This relates to another view of an FST; as a machine that reads one string and generates another.

Morphological Parsing with Finite-State Transducers (Cont…)

- Here's a summary of this four-fold way of thinking about transducers:
  - FST as recognizer: a transducer that takes a pair of strings as input and outputs accept if the string-pair is in the string-pair language, and a *reject* if it is not.
  - FST as generator: a machine that outputs pairs of strings of the language. Thus the output is a yes or no, and a pair of output strings.
  - FST as translator: a machine that reads a string and outputs another string.
  - FST as set relater: a machine that computes relations between sets.

Example of Regular Noun Transition

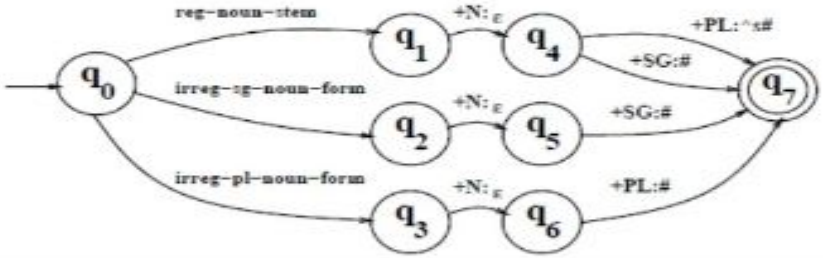


Morphological Parsing with Finite-State Transducers (Cont...)

- We mentioned that for two-level morphology it's convenient to view an FST as having two tapes. The **upper** or **lexical tape**, is composed from characters from the left side of the *a* : *b* pairs; the **lower** or **surface** tape, is composed of characters from the right side of the *a* : *b* pairs.
- Thus each symbol *a* : *b* in the transducer alphabet Σ expresses how the symbol *a* from one tape is mapped to the symbol *b* on the another tape.
- For example *a* : ε means that an *a* on the upper tape will correspond to *nothing* on the lower tape.
- Just as for an FSA, we can write regular expressions in the complex alphabet Σ.
- Since it's most common for symbols to map to themselves, in two-level morphology we call pairs like *a* : *a* **default pairs**, and just refer to them by the single letter *a*.

Morphological Parsing with Finite-State Transducers (Cont...)

- Let's build an FST morphological parser out of our earlier morphotactic FSAs and lexica by adding an extra "lexical" tape and the appropriate morphological features.



A transducer for English nominal number inflection T_{num}. Since both q_1 and q_2 are accepting states, regular nouns can have the plural suffix or not. The morpheme-boundary symbol ^ and word-boundary marker # will be discussed below.

reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e s e	goose
cat	sheep	sheep
dog	m o:i u:ɛ s:e e	mouse
aardvark		

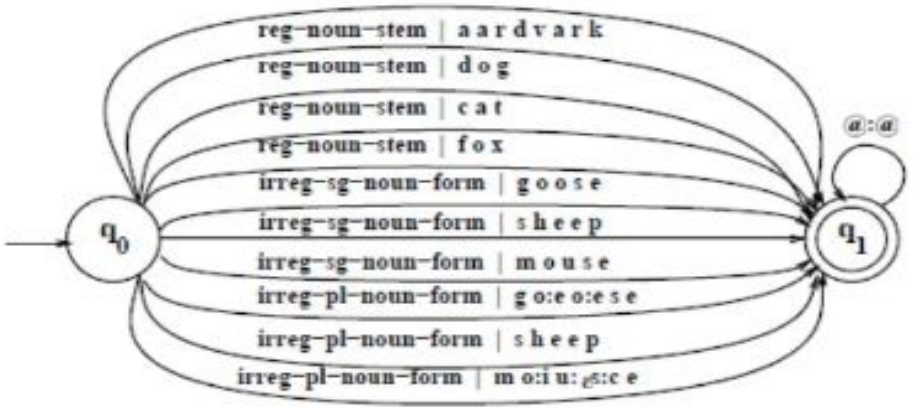
Note that these features map to the empty string ε or the word/morpheme boundary symbol # since there is no segment corresponding to them on the output tape.

Morphological Parsing with Finite-State Transducers (Cont...)

- In order to do this we need to update the lexicon for this transducer, so that irregular plurals like *geese* will parse into the correct stem goose +N +PL.
- We do this by allowing the lexicon to also have two levels.
- Since surface *geese* maps to underlying goose, the new lexical entry will be 'g:g o:e o:e s:s e:e'.
- Regular forms are simpler; the two-level entry for *fox* will now be 'f:f o:o x:x', but by relying on the orthographic convention that f stands for f:f and so on, we can simply refer to it as fox and the form for *geese* as 'g o:e o:e s e'.
- Thus the lexicon will look only slightly more complex:

Morphological Parsing with Finite-State Transducers (Cont…)

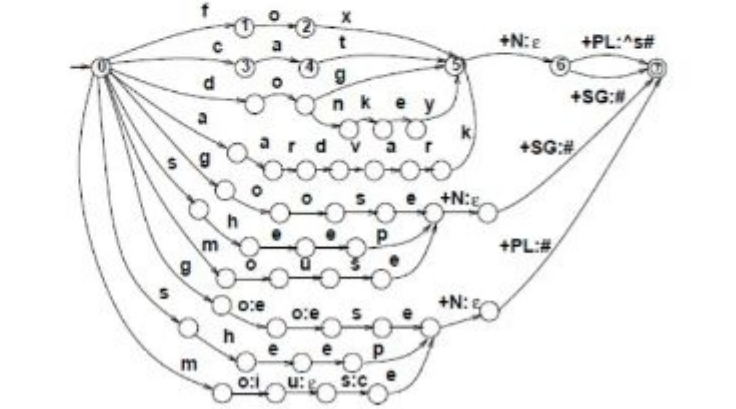
- Our proposed morphological parser needs to map from surface forms like *geese* to lexical forms like goose +N +SG.
- We could do this by **cascading** the lexicon above with the singular/plural automaton.
- Cascading two automata means running them in series with the output of the first feeding the input to the second.
- We would first represent the lexicon of stems in the above table as the FST T_{stems} as follows;



Morphological Parsing with Finite-State Transducers (Cont…)

- Composed Automation
  - Following transducer will map plural nouns into the stem plus the morphological marker +PL, and singular nouns into the stem plus the morpheme +SG.
  - Thus a surface *cats* will map to cat +N +PL as follows: c:c a:a t:t +N:ε +PL:ˆs#

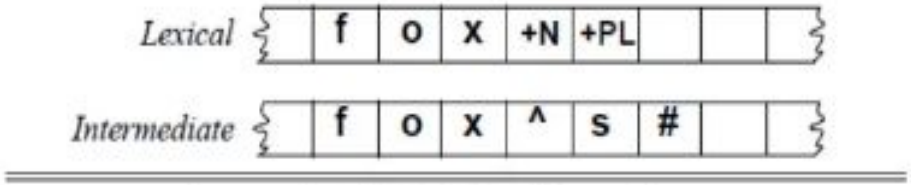
That is, c maps to itself, as do a and t, while the morphological feature +N (recall that this means 'noun') maps to nothing (ε), and the feature +PL (meaning 'plural') maps to ˆs. The symbol ˆ indicates a **morpheme boundary**, while the symbol **#** indicates a **word boundary**,



A fleshed-out English nominal inflection FST $T_{lex} = T_{mun} \circ T_{stems}$

Morphological Parsing with Finite-State Transducers (Cont…)

- It creates intermediate tapes as follows;



An example of the lexical and intermediate tapes.

Orthographic Rules and Finite-State Transducers

- The method described in the previous section will successfully recognize words like *aardvarks* and *mice*.
- But just concatenating the morphemes won't work for cases where there is a spelling change; it would incorrectly reject an input like *foxes* and accept an input like *foxs*.
- We need to deal with the fact that English often requires spelling changes at morpheme boundaries by introducing **spelling rules** (or **orthographic rules**).
- This section introduces a number of notations for writing such rules and shows how to implement the rules as transducers.

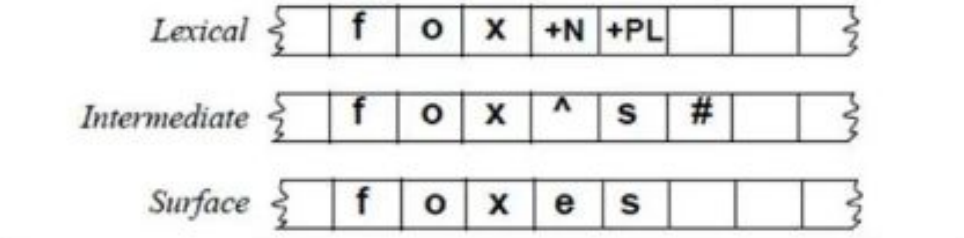
Orthographic Rules and Finite-State Transducers (Cont...)

- Some of these spelling rules:

Name	Description of Rule	Example
Consonant doubling	1-letter consonant doubled before *-ing/-ed*	beg/begging
E deletion	Silent e dropped before *-ing* and *-ed*	make/making
E insertion	e added after *-s,-z,-x,-ch, -sh* before *-s*	watch/watches
Y replacement	*-y* changes to *-ie* before *-s*, *-i* before *-ed*	try/tries
K insertion	verbs ending with *vowel* + *-c* add *-k*	panic/panicked

Orthographic Rules and Finite-State Transducers (Cont…)

- We can think of these spelling changes as taking as input a simple concatenation of morphemes (the 'intermediate output' of the lexical transducer) and producing as output a slightly-modified, (correctly spelled) concatenation of morphemes.
- So for example we could write an E-insertion rule that performs the mapping from the intermediate to surface levels.



An example of the lexical, intermediate and surface tapes. Between each pair of tapes is a 2-level transducer