19CSE453 – Natural Language Processing

Regular Expressions, Finite Automata

By

Ms. Kavitha C.R.

Dept. of Computer Science and Engineering

Amrita School of Computing, Bengaluru

Amrita Vishwa Vidyapeetham

Regular expressions

A regular expression (RE) is used 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.

Properties of Regular Expressions

Followings are some of the important properties of RE –

•RE is a formula in a special language, which can be used for specifying simple classes of strings, a sequence of symbols. In other words, we can say that RE is an algebraic notation for characterizing a set of strings.

•Regular expression requires two things, one is the pattern that we wish to search and other is a corpus of text from which we need to search.

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)**

are also regular expressions.

•If a string is derived from above rules then that would also be a regular expression.

Examples of Regular Expressions

The following table shows a few examples of Regular Expressions −

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, ……..}

RE	Match
/[A-Z]/	an uppercase letter
/[a-z]/	a lowercase letter
/[0-9]/	a single digit

RE	Match
/[wW]oodchuck/	Woodchuck or woodchuck
/[abc]/	‘a’, ‘b’, *or* ‘c’
/[1234567890]/	any digit

RE	Match (single characters)
[^A-Z]	not an uppercase letter
[^Ss]	neither ‘S’ nor ‘s’
[^\.]	not a period
[e^]	either ‘e’ or ‘^’
a^b	the pattern ‘a^b’

RE	Match
woodchucks?	woodchuck or woodchucks
colou?r	color or colour

RE	Match
/beg.n/	any character between *beg* and *n*

RE	Expansion	Match
\d	[0-9]	any digit
\D	[^0-9]	any non-digit
\w	[a-zA-Z0-9_]	any alphanumeric/underscore
\W	[^\w]	a non-alphanumeric
\s	[␣\r␣\t␣\n␣\f␣]	whitespace (space, tab)
\S	[^\s]	Non-whitespace

RE	Match
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	exactly zero or one occurrence of the previous char or expression
{n}	*n* occurrences of the previous char or expression
{n,m}	from *n* to *m* occurrences of the previous char or expression
{n,}	at least *n* occurrences of the previous char or expression

RE	Match
*	an asterisk “*”
\.	a period “.”
\?	a question mark
\n	a newline
\t	a tab

anchors are ^ and $ anchor regular expressions at the beginning and end of the text, respectively

Disjunction of characters inside a regular expression is done with the matching square brackets [ ]. All characters inside [ ] are part of the disjunction

Regular Sets & Their Properties

It may be defined as the set that represents the value of the regular expression and consists specific properties.

Properties of regular sets

- if the union of two regular sets then the resulting set would also be regular.
- If the intersection of two regular sets then the resulting set would also be regular.
- If the complement of regular sets, then the resulting set would also be regular.
- If the difference of two regular sets, then the resulting set would also be regular.
- If the reversal of regular sets, then the resulting set would also be regular.
- If the closure of regular sets, then the resulting set would also be regular.
- If the concatenation of two regular sets, then the resulting set would also be regular.

Finite State Automata (FSA)

An automaton having a finite number of states is called a Finite Automaton (FA) or Finite State automata (FSA).

Mathematically, an automaton can be represented by a 5-tuple (Q, Σ, δ, q0, F),

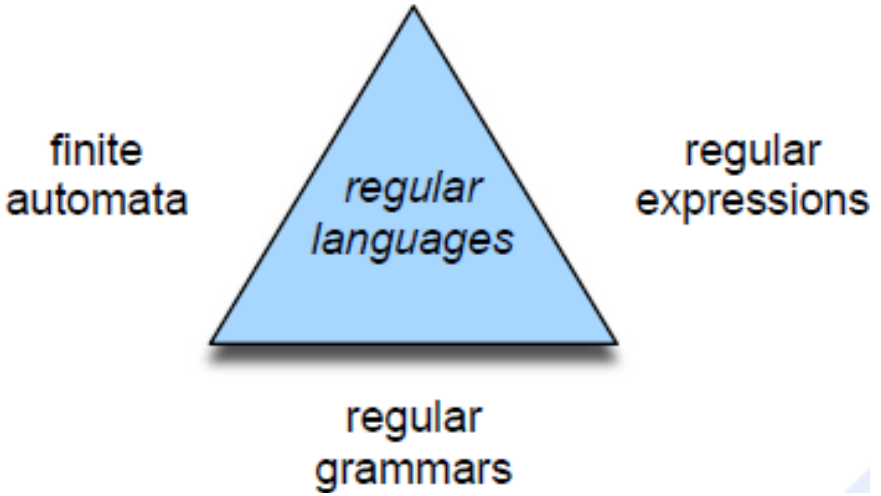
where –

- 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
- 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).

Relation between Finite Automata, Regular Grammars and Regular Expressions

Following points will give us a clear view about the relationship between finite automata, regular grammars and regular expressions −

- finite state automata are the theoretical foundation of computational work and regular expressions is one way of describing them.
- any regular expression can be implemented as FSA and any FSA can be described with a regular expression.
- On the other hand, regular expression is a way to characterize a kind of language called regular language. Hence, regular language can be described with the help of both FSA and regular expression.
- Regular grammar, a formal grammar that can be right-regular or left-regular, is another way to characterize regular lang



Types of Finite State Automata (FSA)

Finite state automation is of two types.

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 $(Q, \Sigma, \delta, q0, F)$,

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 $\delta: Q \times \Sigma \rightarrow Q$.
- $q0$ is the initial state from where any input is processed ($q0 \in Q$).
- F is a set of final state/states of Q ($F \subseteq Q$).

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

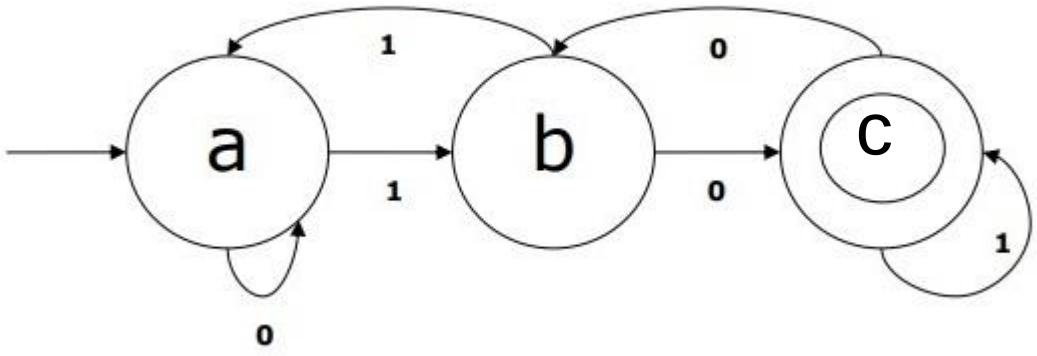
- The states are represented by **vertices**.
- The transitions are shown by labeled **arcs**.
- The initial state is represented by an **empty incoming arc**.
- The final state is represented by **double circle**.

Example of DFA

Suppose a DFA be
- Q = {a, b, c},
- Σ = {0, 1},
- q_0 = {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	c	a
*c	b	c

The graphical representation of this DFA would be as follows −



Non-deterministic Finite Automata (NFA)

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 (NFA).

Mathematically, NFA 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 $\delta: Q \times \Sigma \rightarrow 2^{Q}$.
- q0 is the initial state from where any input is processed ($q0 \in Q$).
- F is a set of final state/states of Q ($F \subseteq Q$).

Whereas graphically (same as DFA), a NFA 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 NFA

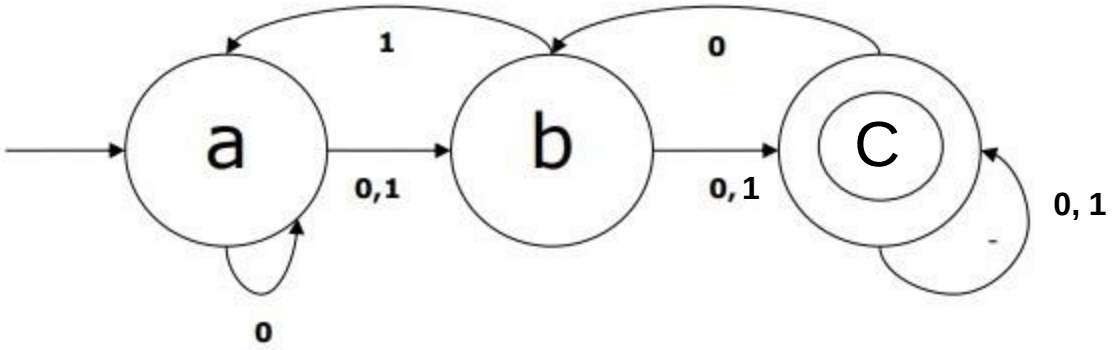
Suppose a NDFA be

- 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 NFA would be as follows −



Regular Languages and FSAs

1. $\emptyset$ is a regular language
2. $\forall a \in \Sigma \cup \epsilon$, $\{a\}$ is a regular language
3. If L_1 and L_2 are regular languages, then so are:
   (a) $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$, the **concatenation** of L_1 and L_2
   (b) $L_1 \cup L_2$, the **union** or **disjunction** of L_1 and L_2
   (c) L_1^*, the **Kleene closure** of L_1

intersection	if L_1 and L_2 are regular languages, then so is $L_1 \cap L_2$, the language consisting of the set of strings that are in both L_1 and L_2.
difference	if L_1 and L_2 are regular languages, then so is $L_1 - L_2$, the language consisting of the set of strings that are in L_1 but not L_2.
complementation	If L_1 is a regular language, then so is $\Sigma^* - L_1$, the set of all possible strings that aren't in L_1.
reversal	If L_1 is a regular language, then so is L_1^R, the language consisting of the set of reversals of all the strings in L_1.

NLP - Word Level Analysis

How can RE be used in NLP?

1) Validate data fields (e.g., dates, email address, URLs, abbreviations)
2) Filter text (e.g., spam, disallowed web sites)
3) Identify particular strings in a text (e.g., token boundaries)
4) Convert the output of one processing component into the format required for a second component

Word Level Analysis - Tokenization

Tokenization is the process of segmenting a string of characters into words.

Depending on the application in hand, you might have to perform ***sentence segmentation*** as well.

Example: I have a can opener; but I can't open these cans.

Word Token

An occurrence of a word

For the above sentence, 11 word tokens.

Word Type

A different realization of a word

For the above sentence, 10 word types.

Sentence Segmentation

Sentence Segmentation

The problem of deciding where the sentences begin and end.

Challenges Involved:

While '!', '?' are quite unambiguous

Period "." is quite ambiguous and can be used additionally for

- **Abbreviations (Dr., Mr., m.p.h.)**
- **Numbers (2.4%, 4.3)**

Approach: build a binary classifier

For each "."

Decides EndOfSentence/NotEndOfSentence

Classifiers can be: hand-written rules, regular expressions, or machine learning

Other Important Features

Case of word with "." : Upper, Lower, Number

Case of word after "." : Upper, Lower, Number

Numeric Features

- **Length of word with "."**
- **Probability (word with "." occurs at end-of-sentence)**
- **Probability (word after "." occurs at beginning-of-sentence)**

Word Tokenization

Issues in Tokenization

Finland's → Finland Finlands Finland's ?

What're, I'm, shouldn't → What are, I am, should not ?

San Francisco → one token or two?

m.p.h. → ??

Handling Hyphenation

Hyphens can be

End-of-Line Hyphen

Used for splitting whole words into part for text justification.
This paper describes MIMIC, an adaptive mixed initia-tive spoken dialogue system that provides movie show-time information.

Lexical Hyphen

Certain prefixes are often written hyphenated, e.g. co-, pre-, meta-, multi-, etc.

Normalization

Why to "normalize"?

Indexed text and query terms must have the same form.

U.S.A. and USA should be matched

We implicitly define equivalence classes of terms

Three forms of Normalization

- *Case folding*
- *Stemming*
- *Lemmatization*

Case Folding

Reduce all letters to lower case

Possible exceptions (Task dependent):

- Upper case in mid sentence, may point to named entities (e.g. General Motors)
- Words conveyed in CAPS mean a strong conveyance.

Lemmatization

Reduce inflections or variant forms to base form:

- am, are, is → be
- car, cars, car's, cars' → car

Have to find the correct dictionary headword form

Stemming

Reducing terms to their stems, used in information retrieval Crude chopping of affixes

- language dependent
- *automate(s), automatic, automation* all reduced to *automat*

for example compressed and compression are both accepted as equivalent to compress.

Morphology

Morphology studies the internal structure of words, how words are built up from smaller meaningful units called morphemes

dogs

2 morphemes, ‘dog’ and ‘s’

‘s’ is a plural marker on nouns

unladylike

3 morphemes

un- ‘not’

lady ‘well-behaved woman’

-like ‘having the characteristic of’

Morphology

Allomorphs

Variants of the same morpheme, but cannot be replaced by one another

Example

opposite: un-happy, in-comprehensible, im-possible, ir-rational

Bound and Free Morphemes

Bound

Cannot appear as a word by itself.

-s (dog-s), -ly (quick-ly), -ed (walk-ed)

Free

Can appear as a word by itself; often can combine with other morphemes too.

house (house-s), walk (walk-ed), of, the, or

Morphological Parsing

The term morphological parsing is related to the parsing of morphemes.
Morphological parsing is 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*.
the word *foxes*, is made up of two morphemes, one is *fox* and other is -*es*.

In other sense, 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

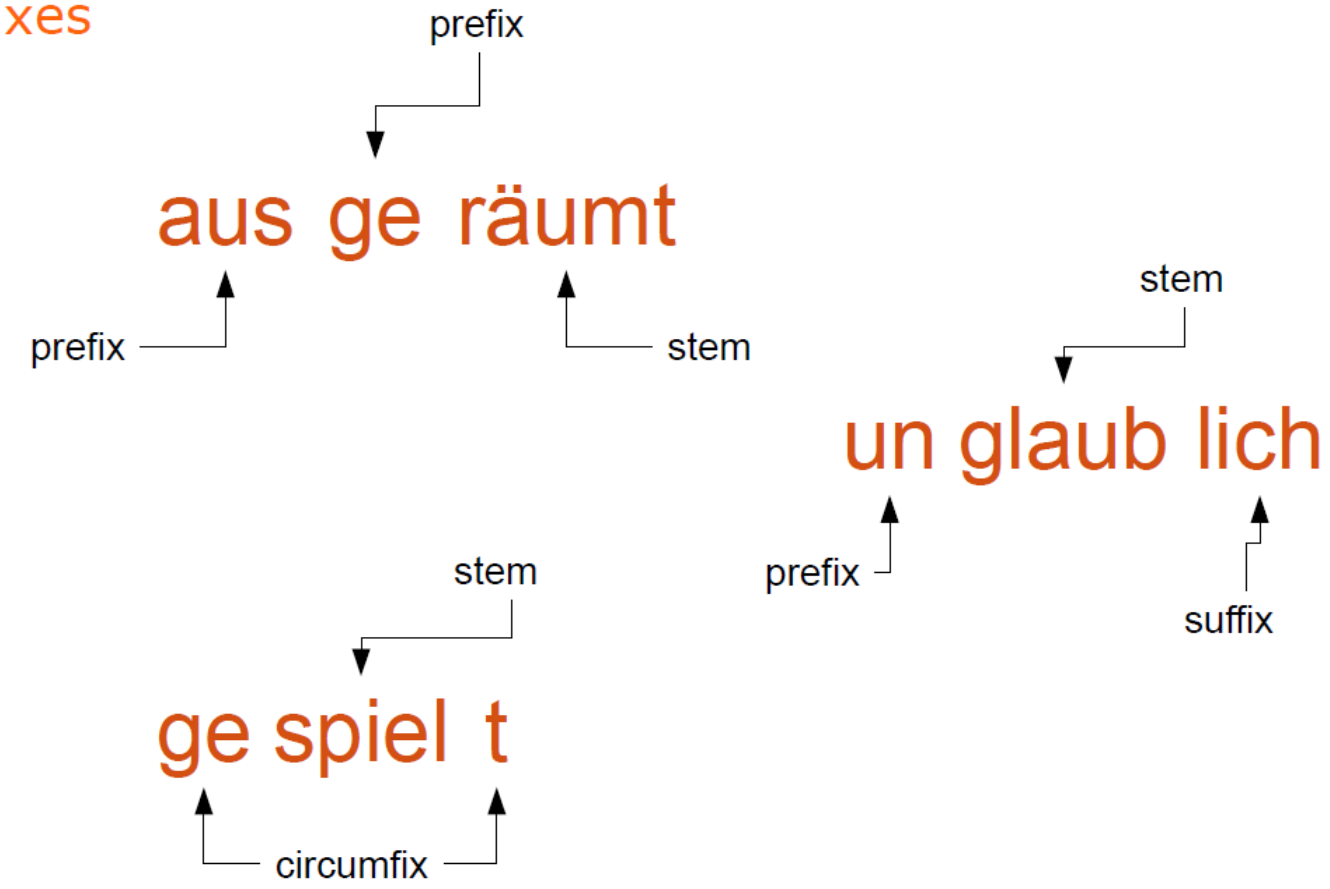
It is the core meaningful unit of a word. We can also say that it is the root of the word.
For example, in the word foxes, the stem is fox.

•**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** − prefixes precede the stem. For example, in the word unbuckle, un is the prefix.
- **Suffixes** − suffixes follow the stem. For example, in the word cats, -s is the suffix.
- **Infixes** − 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.

Examples

Affixes



- Stem: main morpheme
- Affixes: additional meanings
  - Prefix (before)
  - Suffix (after)
  - Circumfix (both)
  - Infix (middle)

Mostly, stems are free morphemes and affixes are bound morphemes

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

lexicon includes the list of stems and affixes along with the basic information about them.

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.

Example: the morphotactic fact is that the English plural morpheme always follows the noun rather than preceeding it.

Orthographic rules

The spelling rules are used to model the changes occurring in a word.

Example: the rule of converting y to ie in word like city+s = cities and not citys.

Content and Functional morphemes

Content morphemes

Carry some semantic content

-able, un-

Functional morphemes

Provide grammatical information

-s (plural), -Is (3rd person singular of "to be" noun)

Combining Morphemes

1. Inflection

- Inflection
  - Resulting word is of the same class of the original word
  - Plurals, conjugation of verbs
  - *cat (cats), play(played)*
- Derivation
- Compounding
- Cliticization

	Regular Nouns		Irregular Nouns	
Singular	cat	thrush	mouse	ox
Plural	cats	thrushes	mice	oxen

Morphological Class	Regularly Inflected Verbs			
stem	walk	merge	try	map
-s form	walks	merges	tries	maps
-ing participle	walking	merging	trying	mapping
Past form or *-ed* participle	walked	merged	tried	mapped

Morphological Class	Irregularly Inflected Verbs		
stem	eat	catch	cut
-s form	eats	catches	cuts
-ing participle	eating	catching	cutting
preterite	ate	caught	cut
past participle	eaten	caught	cut

Inflectional morphology

Grammatical: number, tense, case, gender

Creates new forms of the same word:

bring, brought, brings, bringing

2. Derivation

- Inflection
- Derivation
  - Resulting word is of a distinct class of the original word
  - For instance, verbs to nouns or to adjectives
  - *compute (computation, computational)*
- Compounding
- Cliticization

Suffix	Base Verb/Adjective	Derived Noun
-ation	computerize (V)	computerization
-ee	appoint (V)	appointee
-er	kill (V)	killer
-ness	fuzzy (A)	fuzziness

Suffix	Base Noun/Verb	Derived Adjective
-al	computation (N)	computational
-able	embrace (V)	embraceable
-less	clue (N)	clueless

Derivational morphology

Creates new words by changing part-of-speech:

logic, logical, illogical, illogicality, logician

Fairly systematic but some derivations missing:

sincere - sincerity, scarce - scarcity, curious - curiosity, fierce - fiercity?

3. Compounding

- Inflection
- Derivation
- Compounding
  - Combination of multiple word stems
  - *doghouse*
- Cliticization

Compounding

Words formed by combining two or more words

Example in English:

Adj + Adj → Adj: bitter-sweet

N + N → N: rain-bow

V + N → V: pick-pocket

P + V → V: over-do

4. Cliticization

- Inflection
- Derivation
- Compounding
- Cliticization
  - Combination of a word stem with a clitic
  - Clitic: a morpheme that acts like a word but is reduced and attached to another word
  - *I've, l'opera*

Full Form	Clitic	Full Form	Clitic
am	'm	have	've
are	're	has	's
is	's	had	'd
will	'll	would	'd

	proclitic	proclitic	stem	affix	enclitic
Arabic	w	b	Hsn	At	hm
Gloss	and	by	virtue	s	their

Morphological Processes

Concatenation

Adding continuous affixes - the most common process:

hope+less, un+happy, anti+capital+ist+s

Often, there are phonological/graphemic changes on morpheme boundaries:

book + s [s],

shoe + s [z]

happy +er → happier

Morphological Processes

Reduplication: part of the word or the entire word is doubled

Nama: ‘go’ (look), ‘go-go’ (examine with attention)

Tagalog: ‘basa’ (read), ‘ba-basa’(will read)

Sanskrit: ‘pac’ (cook), ‘papa¯ca’ (perfect form, cooked)

Phrasal reduplication (Telugu): *pillava¯d. u nad.ustu¯ nad.ustu¯ pad.i po¯ya¯d. u*
(The child fell down while walking)

Suppletion

‘irregular’ relation between the words

go - went, good - better

Morpheme internal changes

The word changes internally

sing - sang - sung, man - men, goose - geese

Word Formation

Compounding

Words formed by combining two or more words

Example in English:

Adj + Adj → Adj: bitter-sweet

N + N → N: rain-bow

V + N → V: pick-pocket

P + V → V: over-do

Acronyms

laser: **Light Amplification by Simulated Emission of Radiation**

Word Formation

Blending

Parts of two different words are combined

breakfast + lunch → brunch

smoke + fog → smog

motor + hotel → motel

Clipping

Longer words are shortened

doctor, laboratory, advertisement, dormitory, examination, bicycle, refrigerator

Morphological Processing

Lemmatization: word → lemma

saw → {see, saw}

Morphological analysis : word → setOf(lemma +tag)

saw → { <see, verb.past>, < saw, noun.sg>}

Tagging: word → tag, considers context

Peter *saw* her → { <see, verb.past>}

Morpheme segmentation: de-nation-al-iz-ation

Generation: see + verb.past → saw

Applications

Text-to-speech synthesis:

lead: **verb or noun?**
read: **present or past?**

Search and information retrieval ,

Machine translation,

grammar correction

Morphological Analysis

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)

Present participle

Past participle

Goal

To take input forms like those in the first column and produce output forms like those in the second column.

Output contains stem and additional information;

+N for noun, +SG for singular, +PL for plural, +V for verb etc.

Issues involved

boy → boys
fly → flys → flies (y→ i rule)

Toiling → toil
Duckling → duckl?

Getter → get + er
Doer → do + er
Beer → be + er?

Knowledge Required

Knowledge of stems or roots

Duck is a possible root, not *duckl*.

We need a dictionary (lexicon)

Morphotactics

Which class of morphemes follow other classes of morphemes inside the word?

Ex: plural morpheme follows the noun

Spelling change rules

Adjust the surface form using spelling

change rules Get + er → getter

Morphological Parsing

- Breaking down words into components and building a structured representation
  - English:
    - cats → cat +N +Pl
    - caught → catch +V +Past
  - Spanish:
    - vino (came) → venir +V +Perf +3P +Sg
    - vino (wine) → vino +N +Masc +Sg

Motivation for morphological parsing

- Information retrieval
  - Normalize verb tenses, plurals, grammar cases
- Machine translation
  - Translation based on the stem

Morphological Parsing

- Resources
  - Lexicon
    - List of all stems and affixes
  - Morphotactics
    - A model of morpheme ordering in a word
    - e.g., plurals are suffixes in English
  - Ortographic rules
    - Rules for changing in the words when combining morphemes
    - e.g., city → cities

Stemming Vs Lemmatization

- Stemming: stripping off word endings (rule-based)
  - foxes → fox
  - going → go

- Lemmatization: mapping the word to its lemma (lexicon-based)
  - sang, sung → sing
  - going, went, goes → go

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

Stemming Vs Lemmatization

- ***Stemming algorithms*** *work by cutting off the* ***end or the beginning*** *of the word, taking into account a list of common prefixes and suffixes that can be found in an inflected word. This indiscriminate cutting can be successful in some occasions, but not always, and that is why we affirm that this approach presents some limitations.*

  studies studi

  studying study

- ***Lemmatization****, on the other hand, takes into consideration the* ***morphological analysis*** *of the words. To do so, it is necessary to have* ***detailed dictionaries*** *which the algorithm can look through to link the form back to its lemma. Better results than stemming*

1. stemming might return just *s for word saw,* whereas lemmatization would attempt to return either *see* or *saw* depending on whether the use of the token was as a verb or a noun

2. stemming most commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma

3. The most common algorithm for stemming English, and one that has repeatedly been shown to be empirically very effective, is *Porter's algorithm* (Porter, 1980).

Porter's algorithm consists of 5 phases of word reductions, applied sequentially. Within each phase there are various conventions to select rules, such as selecting the rule from each rule group that applies to the longest suffix.

In the first phase, this convention is used with the following rule group:

(F)	Rule			Example		
	SSES	→	SS	caresses	→	caress
	IES	→	I	ponies	→	poni
	SS	→	SS	caress	→	caress
	S	→		cats	→	cat

Many of the later rules use a concept of the *measure* of a word, which loosely checks the number of syllables to see whether a word is long enough that it is reasonable to regard the matching portion of a rule as a suffix rather than as part of the stem of a word. For example, the rule:

$m > 1$

ment

would map *replacement* to *replac*, but not *cement* to *c*.

Official site of Porter Stemmer is:

http://www.tartarus.org/~martin/PorterStemmer/

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

A comparison of three stemming algorithms on a sample text

http://www.cs.waikato.ac.nz/~eibe/stemmers/
http://www.comp.lancs.ac.uk/computing/research/stemming/

Stemmers use language-specific rules, but they require less knowledge than a lemmatizer, which needs a complete vocabulary and morphological analysis to correctly lemmatize words. Particular domains may also require special stemming rules. However, the exact stemmed form does not matter, only the equivalence classes it forms.

Rather than using a stemmer, you can use a *lemmatizer*, a tool from Natural Language Processing which does full morphological analysis to accurately identify the lemma for each word. Doing full morphological analysis produces at most very modest benefits for retrieval.

It is hard to say more, because either form of normalization tends not to improve English information retrieval performance in aggregate - at least not by very much. While it helps a lot for some queries, it equally hurts performance a lot for others.

Stemming increases recall while harming precision.

As an example of what can go wrong, note that the Porter stemmer stems all of the following words:

operate operating operates operation operative operatives operational

to oper

However, since *operate* in its various forms is a common verb, we would expect to lose considerable precision on queries such as the following with Porter stemming:

operational and research

operating and system

operative and dentistry

For a case like this, moving to using a lemmatizer would not completely fix the problem because particular inflectional forms are used in particular collocations:

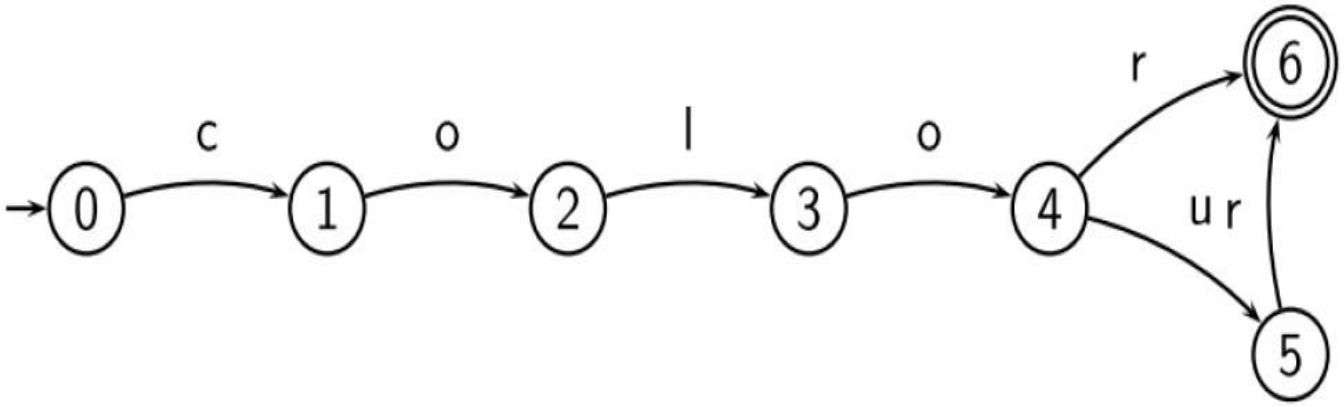
a sentence with the words *operate* and *system* is not a good match for the query operating and system

Getting better value from term normalization depends more on pragmatic issues of word use than on formal issues of linguistic morphology

Snowball stemmer: This algorithm is also known as the Porter2 stemming algorithm. It is almost universally accepted as better than the Porter stemmer

Finite-state methods for morphology

Finite State Automaton (FSA)



What is FSA?

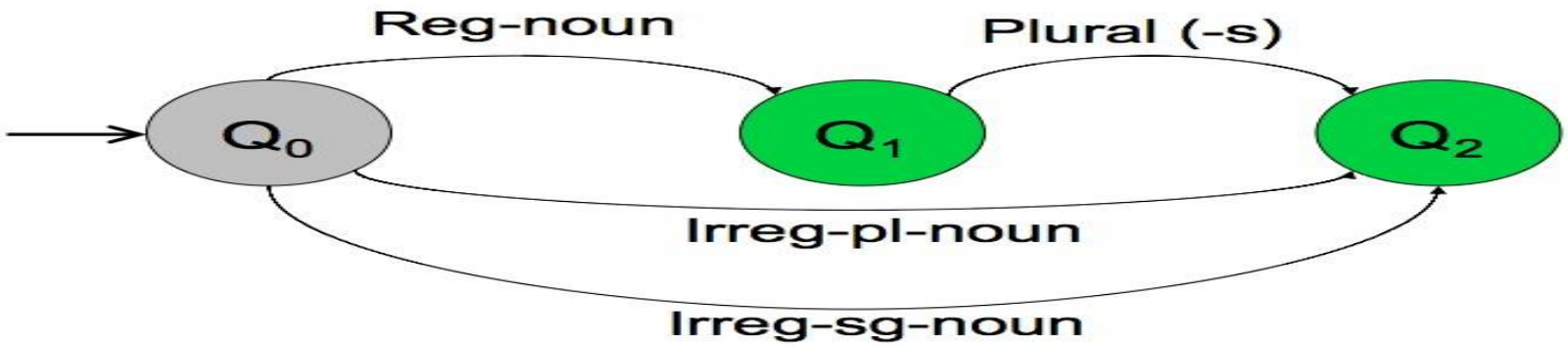
A kind of directed graph

Nodes are called states, edges are labeled with symbols

Start state and accepting states

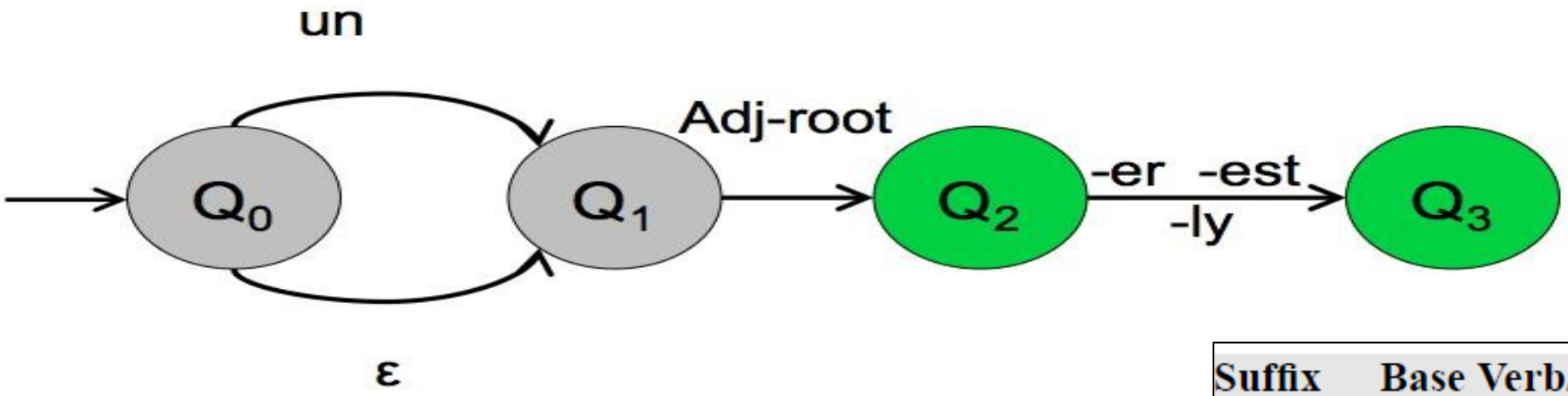
Recognizes regular languages, i.e., languages specified by regular expressions

FSA for nominal inflection in English



	Regular Nouns		Irregular Nouns	
Singular	cat	thrush	mouse	ox
Plural	cats	thrushes	mice	oxen

FSA for English Adjectives



Suffix	Base Verb/Adjective	Derived Noun
-ation	computerize (V)	computerization
-ee	appoint (V)	appointee
-er	kill (V)	killer
-ness	fuzzy (A)	fuzziness

Word modeled

happy, happier, happiest, real, unreal, cool, coolly,

clear, clearly, unclear, unclearly, ...

Morphotactics

The last two examples model some parts of the English

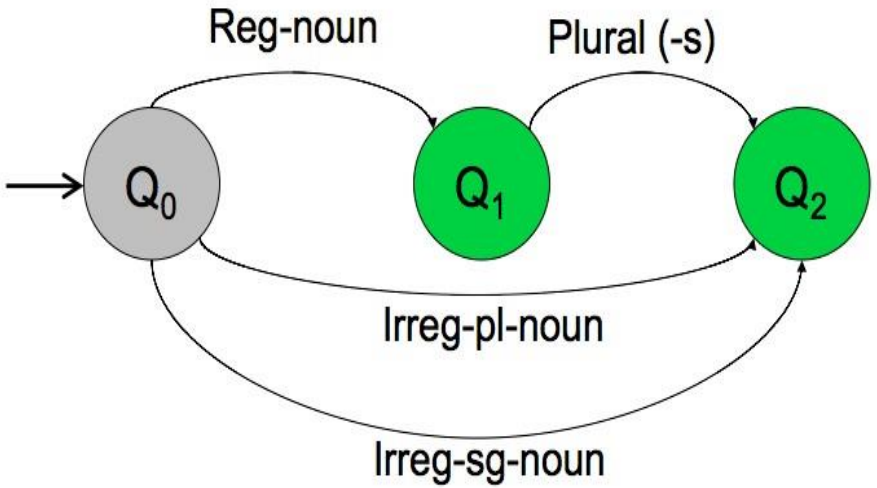
morphotactics

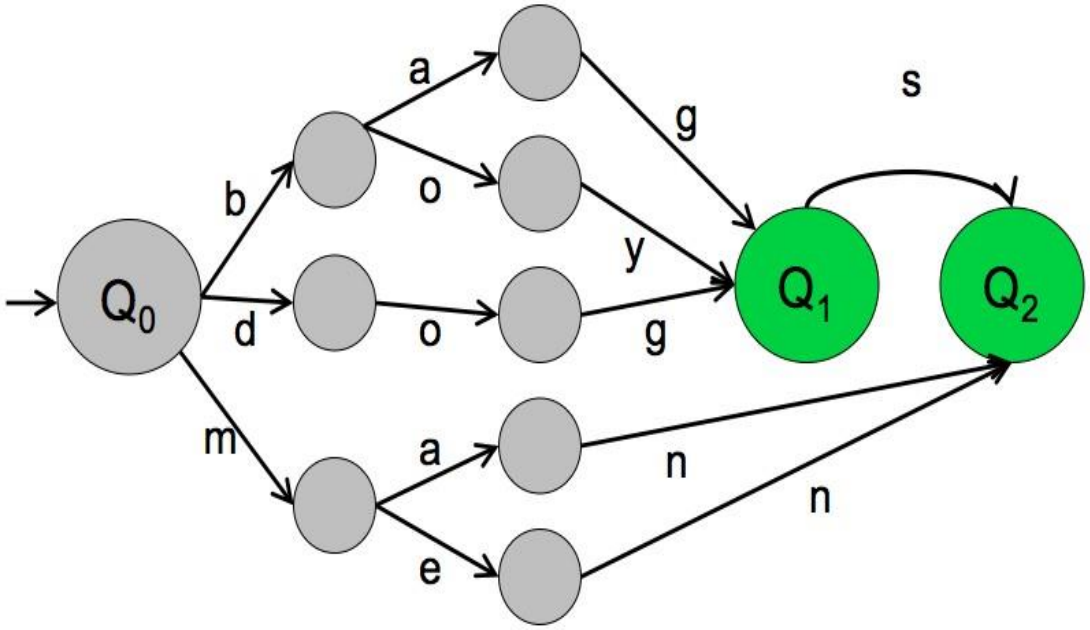
But what about the information about regular and irregular roots?

Lexicon

Can we include the lexicon in the FSA?

FSA for nominal inflection in English

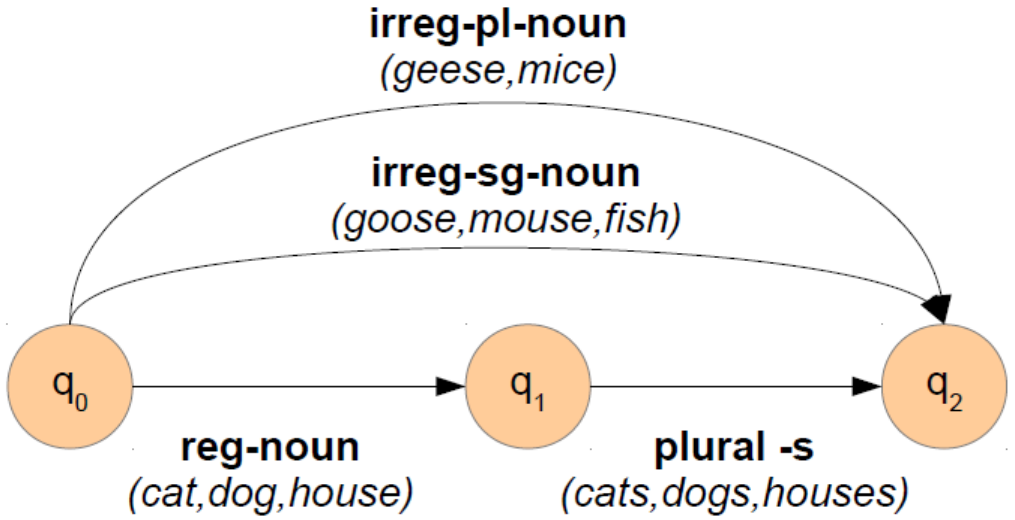


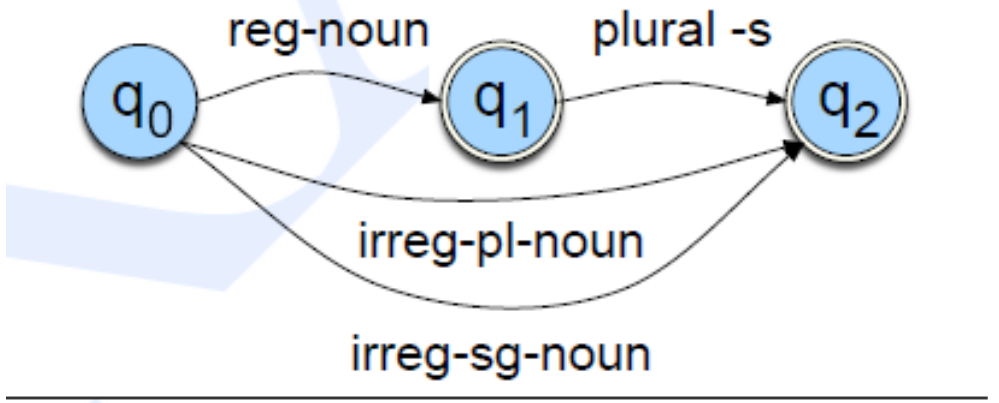


After adding a mini-lexicon

Finite-State Lexicon

- FSA for English nominal inflection (same word category)





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

Finite-State Lexicon

- FSA for derivational morphology (distinct word categories)

Correct adjectives:
- cooler
- unhappiest
- bigger

Incorrect adjectives:
- unbigger
- oranger
- smally

Solution: classes of roots (**adj-root$_1$**, **adj-root$_2$**, etc.)

Some properties of FSA

- Recognizing problem can be solved in linear time (independent of the size of the automaton)
- There is an algorithm to transform each automaton into a unique equivalent automaton with the least number of states
- An FSA is deterministic iff it has no empty (ϵ) transition and for each state and each symbol, there is at most one applicable transition
- Every non-deterministic automaton can be transformed into a deterministic one

Finite State Transducers (FST)

FSAs are language recognizers/generators.

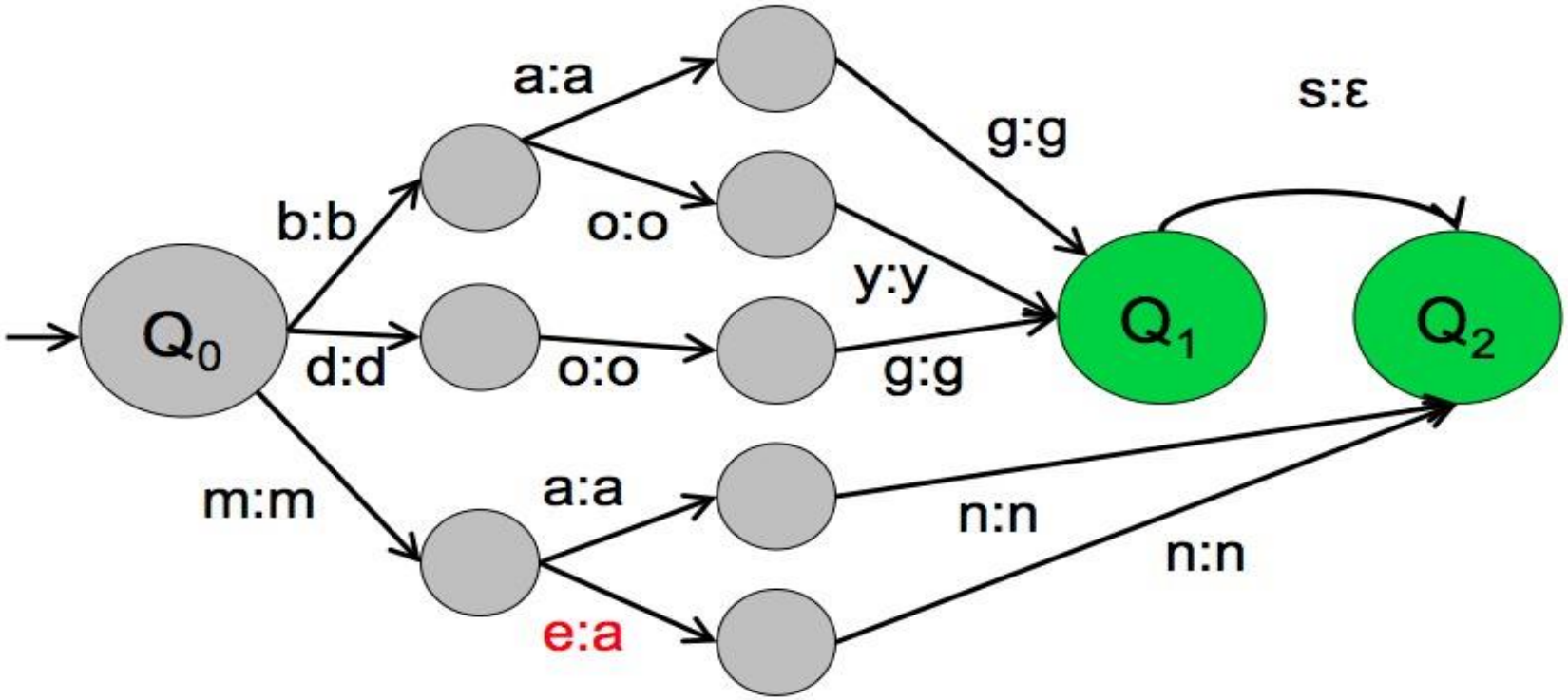
We need transducers to build Morphological Analyzers

Finite State Transducers

Translate strings from one language to strings in another language

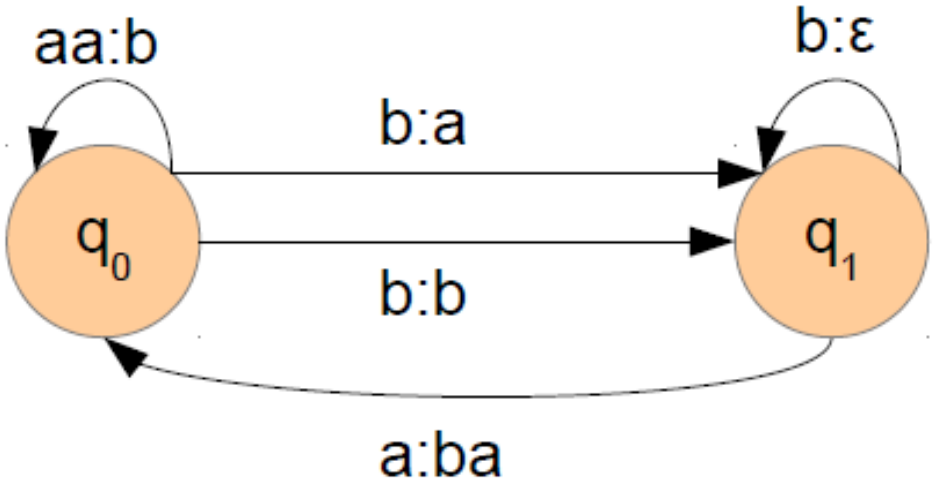
Like FSA, but each edge is associated with two strings

An Example of *Finite State Transducers (FST)*



Finite-State Transducers (FST)

- FST is a type of FSA which maps between two sets of symbols.
- It is a two-tape automaton that recognizes or generates pairs of strings, one from each type.
- FST defines relations between sets of strings.



FST for NLP

- FST as recognizer
  - Takes a pair of strings and accepts or rejects them
- FST as generator
  - Outputs a pair of strings for a language
- FST as translator
  - Reads a string and outputs another string
  - Morphological parsing: letters (input); morphemes (output)
- FST as relater
  - Computes relations between sets

- **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 *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.

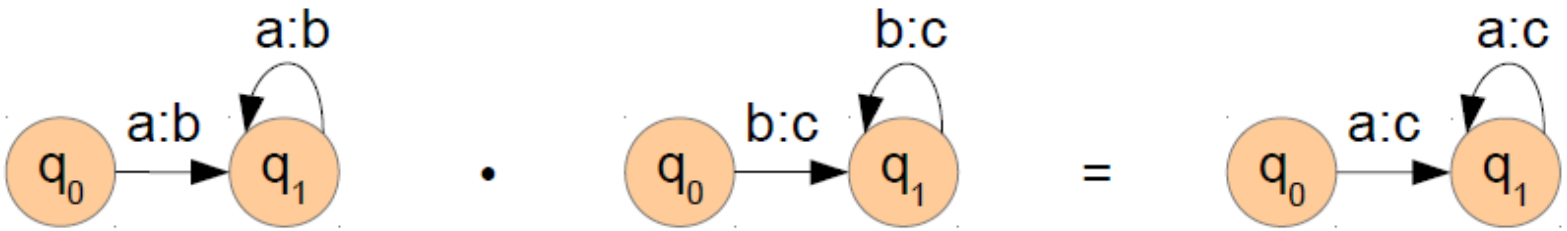
FST – Formal Definition

- $Q = q_0 q_1 q_2 \ldots q_{N-1}$: finite set of N states
- Σ : finite input alphabet of symbols
- Δ : finite output alphabet of symbols
- q_0 : the start state
- F: the set of final states, $F \subseteq Q$
- $\delta(q, w)$: the transition function ($q \in Q$; input string $w \in \Sigma$) returns a set of new states $Q' \in Q$
- $\sigma(q, w)$: the output function ($q \in Q$; input string $w \in \Sigma$) returns a set of output strings $o \in \Delta$

Regular Languages and Regular Relations

- FSA: Regular languages are sets of strings

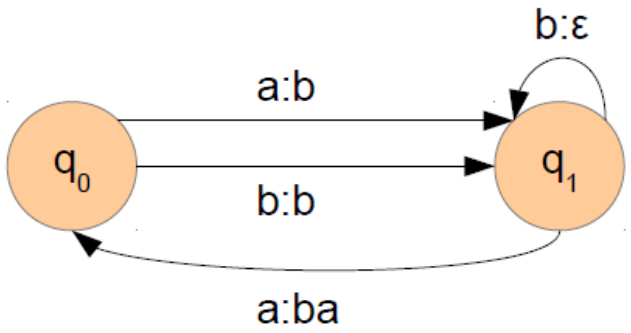
- FST: Regular relations are sets of pairs of strings
  - Properties
    - Inversion: It switches the input and output labels
    - Composition:



The composition of [a:b]+ with [b:c]+ to produce [a:c]+

Deterministic FST

- Not all FST can be determinized: They require search algorithms
- Sequential transducers: deterministic subtype of FST



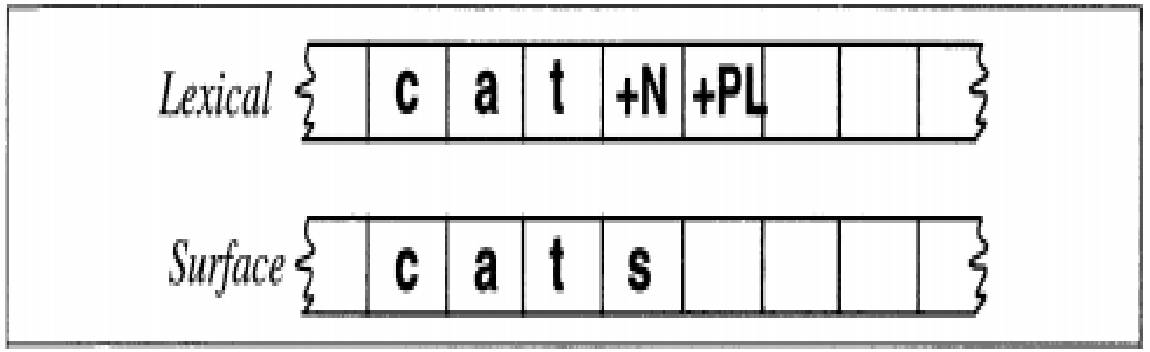
- Subsequencial transducers
  - Generate an additional output string at the final states

Determinism for FST

- Properties of sequential and subsequential transducers
  - Efficient
  - Linear in their input length
  - There are efficient algorithms for their determinization
  - But cannot handle ambiguity

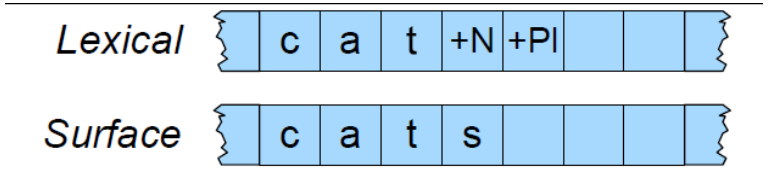
Two-level Morphology

- Given the input *cats*, we would like to output *cat+N+PL*, telling us that cats a plural noun.
- We do this via a version of **two-level morphology**, a correspondence between a lexical level (morphemes and features) to a surface level (actual spelling).



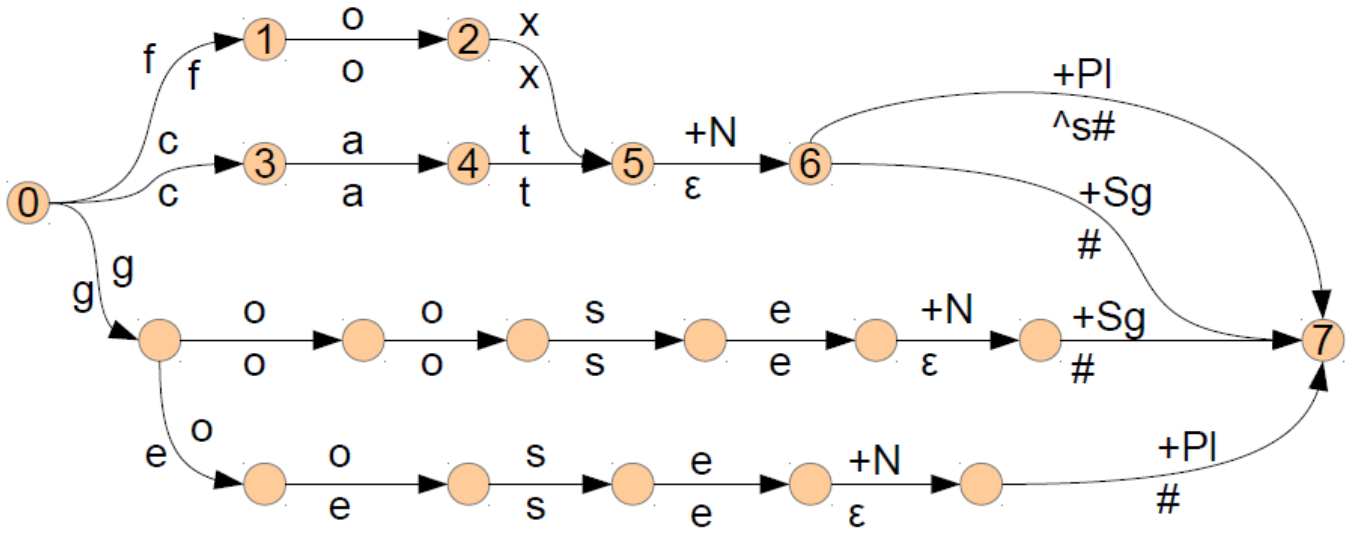
FST for Morphological Parsing

- Two tapes
  - Upper (lexical) tape: input alphabet Σ
    - cat +N +Pl
  - Lower (surface) tape: output alphabet Δ
    - cats



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

- goose/geese: g:g o:e o:e s:s e:e
  - Feasible pairs (e.g., o:e) vs. default pairs (g:g)



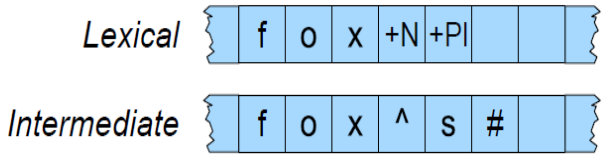
 # symbol is a special symbol which marks the word boundary

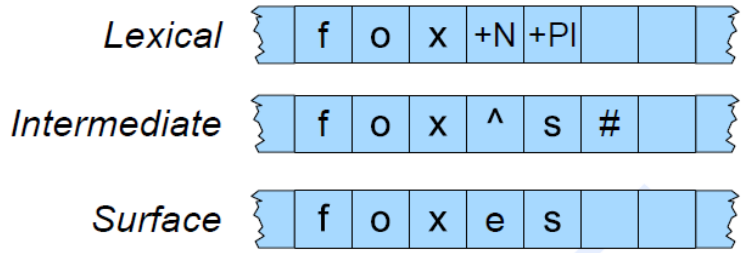
^ denotes morpheme boundary

Intermediate tape for Spelling change rules

- Plural of „fox" is „foxes" not „foxs"

- Consonant double: beg/begging
- E deletion: make/making
- E insertion: watch/watches
- Y replacement: try/tries
- K insertion: panic/panicked

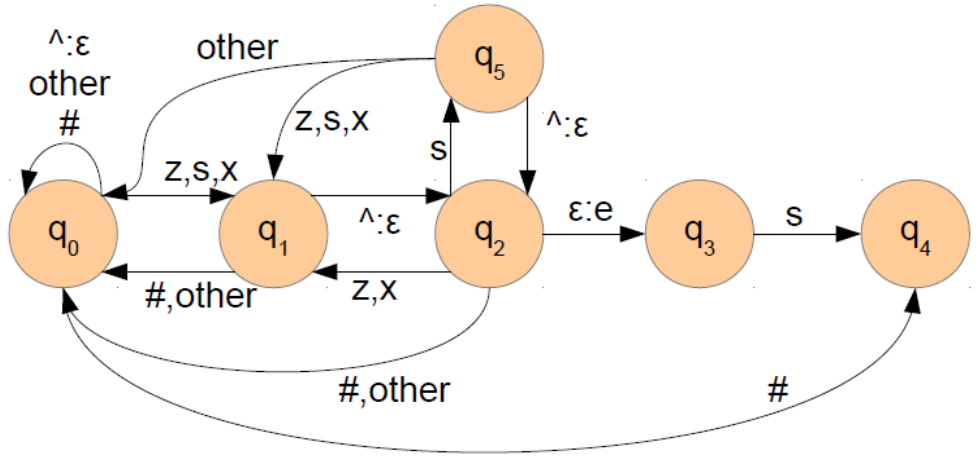




A spelling change rule would insert an e only in the appropriate environment.

FST and Ortographical Rules

- Lexical: foxes +N +Pl
- Intermediate: fox^s#
- Surface: foxes

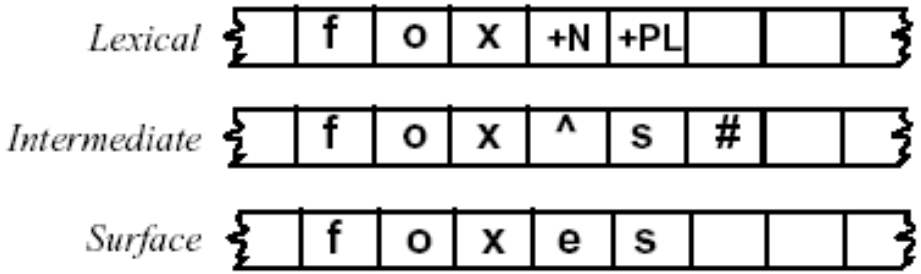


Name	Description of Rule	Example
Consonant doubling	1-letter consonant doubled before *-ing/-ed*	beg/begging
E deletion	Silent e dropped before *-ing* and *-ed*	make/making
E insertion	e added after *-s,-z,-x,-ch, -sh* before *-s*	watch/watches
Y replacement	*-y* changes to *-ie* before *-s*, *-i* before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

Spelling rules (or orthographic 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	Verb ending with ***vowel*** + ***-c*** add ***-k***	panic/panicked

– **These spelling changes can be thought as taking as input a simple concatenation of morphemes and producing as output a slightly-modified concatenation of morphemes.**



- "insert an *e* on the surface tape just when the lexical tape has a morpheme ending in *x* (or *z*, etc) and the next morphemes is *-s*"

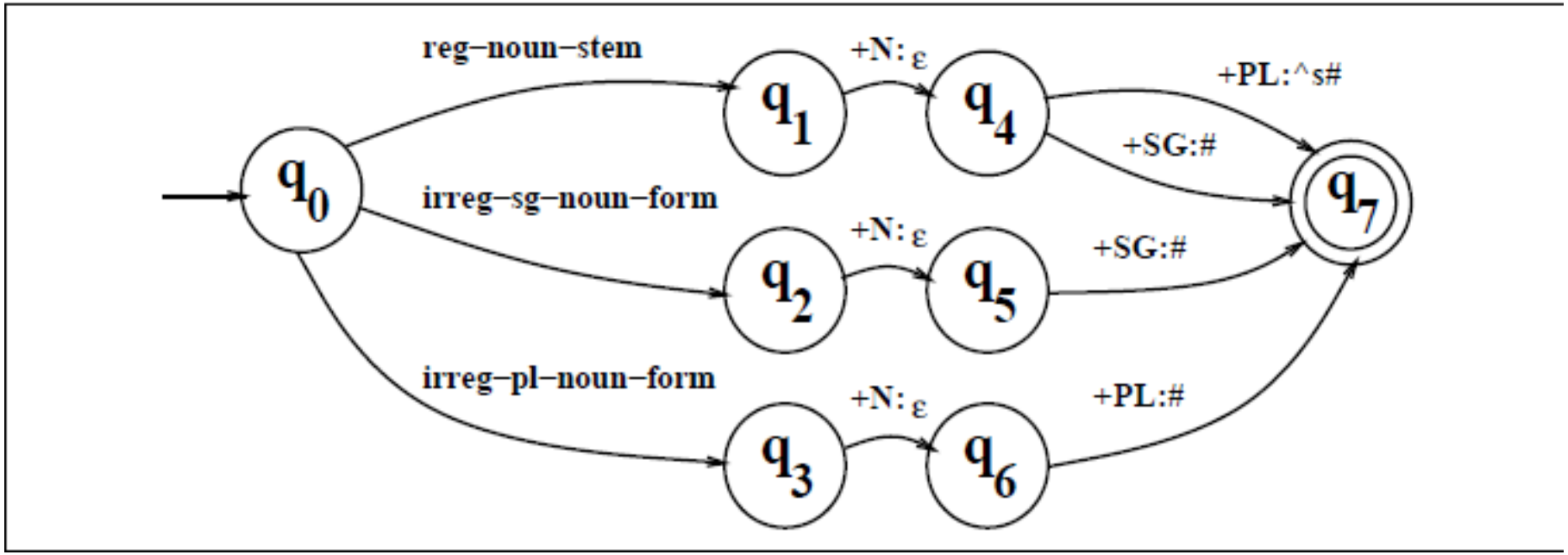
$$\epsilon \rightarrow \text{e} / \left\{ \begin{matrix} x \\ s \\ z \end{matrix} \right\} \hat{} \_\_\_ \text{s\#}$$

^ denotes morpheme boundary

"rewrite a as b when it occurs between c and d" can be written as

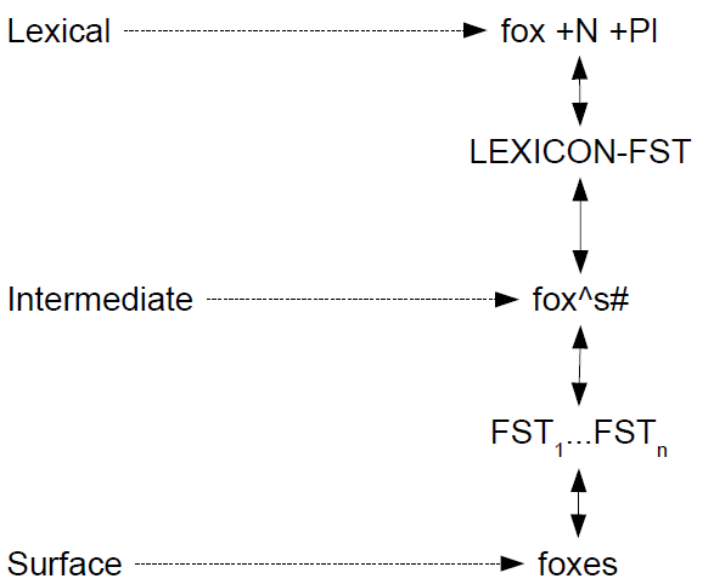
a → b / c___d

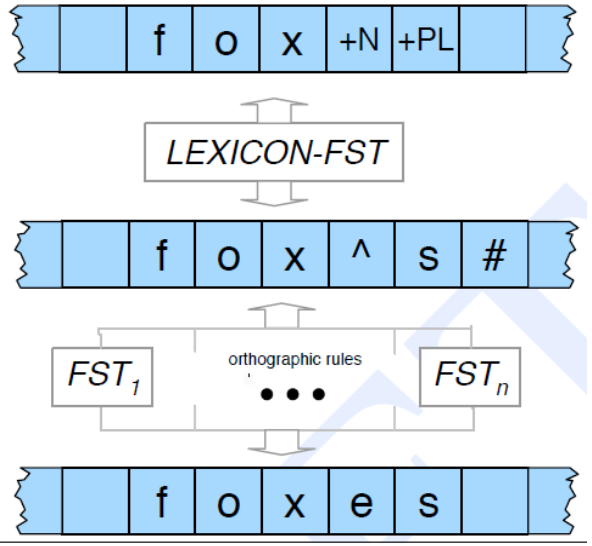
A transducer for English nominal number inflection

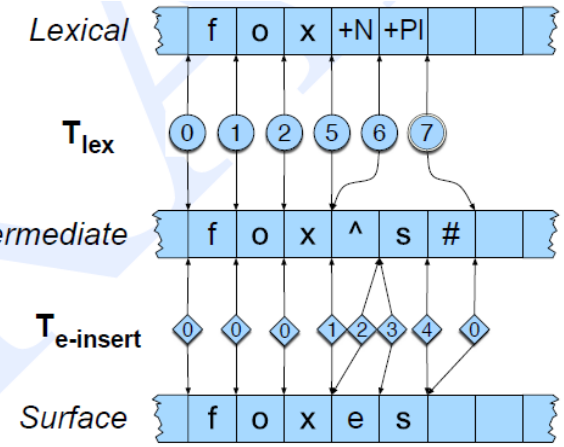


reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e s e	goose
cat	sheep	sheep
dog	m o:i u:ɛ s:c e	mouse
aardvark		

Combination of FST Lexicon and Rules







FST Lexicon and Rules

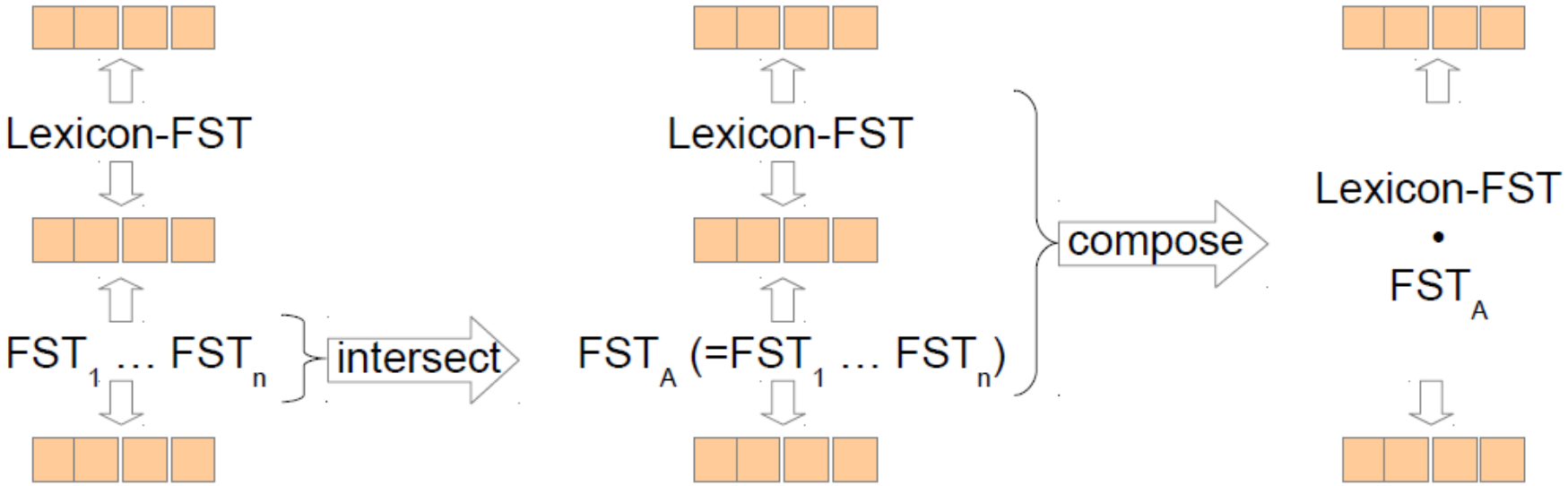
- Disambiguation
  - For some cases, it requires external evidences:
    - I saw two foxes yesterday. (fox +N +Pl)
    - That trickster foxes me every time! (fox +V +3SG)
  - But it can handle local ambiguity (intersection & composition)
    - „asses" vs. „assess"

Combination of FST Lexicon and Rules

FST Lexicon and Rules

- Intersection & Composition

Intersection is the cartesian product of states



Porter Stemmer (Lexicon-Free FST)

- Popular for information retrieval and text categorization tasks
- It is based on a series of simple cascade rules
  - ATIONAL → ATE (relational → relate)
  - ING → ε (motoring → motor)
  - SSES → SS (grasses → grass)
- But it commits many errors:
  - ORGANIZATION → ORGAN
  - DOING → DOE

Errors of Commission		Errors of Omission	
organization	organ	European	Europe
doing	doe	analysis	analyzes
numerical	numerous	noise	noisy
policy	police	sparse	sparsity

Further reading

- Book Jurafski & Martin
  - Chapters 2 and 3 (until section 3.8)
- Regular expressions
  - http://www.regular-expressions.info/

Thank You