Words and Transducers

- Orthographic and Morphological rules,
- Survey of English morphology,
  - Prefixes, suffixes,
  - Infixes, circumfixes,
  - inflection, derivation,
  - compounding, cliticization.
- Finite-state Morphological parsing,
  - lexicon, morphotactics,
  - orthographic rules,
- Building a finite-state Lexicon,
  - Working for words,
  - Reg/Irreg noun,
  - Reg/Irreg verb,
- Finite state Transducers,
  - Working for String/set of strings,
  - FST as recognizer,
  - FST as generator,
  - FST as translator,
  - FST as set relater.
- Sequential transducers and determinism,
- FSTs for Morphological parsing,
- Transducers and Orthographic rules,
- Combining FST Lexicon and Rules,
- Lexicon-free FSTs: The Porter Stemmer,
- Word and Sentence Tokenization,
- Minimum Edit Distance.

1. Words and Transducers (Some Concepts)

- Plural e.g., woodchucks was easy to search these type of plurals just tacks an s on to the end. (e.g., using disjunctions or Pipe Symbol And Paranthesis)
- Consider words like **FOX**, and a **FISH**, and **PUCARRY** a soft-drink.
- Hunting for the plurals of these words takes more than just tacking on an **S**.
- The plural of
  - **fox** is **foxes**;
  - of **pucarry** is **pucarries**;
  - and of **goose** is **geese**.
- Further, **fish don't usually change their form when they are plural**

1. Words and Transducers (Some Concepts) (Cont..)

- It takes two kinds of knowledge to correctly search for singulars and plurals of these forms/

  (1) **Orthographic rules** tell us that English words ending in -y are pluralized by changing the -y to -i- and adding an -es.

  (2)**Morphological rules** tell us that

  - **fish** has a null plural, and that
  - the plural of **goose** is formed by changing the vowel.

- Recognizing that a word **foxes** breaks down into **component morphemes (fox and -es)** and building a structured representation of this fact is called **morphological parsing**
- Parsing means taking an input and producing some sort of linguistic structure for it

1. Words and Transducers (Some Concepts) (Cont..)

- To solve the morphological parsing problem, why couldn't we just store all the plural forms of English nouns and -ing forms of English verbs in a dictionary and do parsing by lookup? Sometimes we can do this

For example; for English speech recognition this is exactly what we do.

- **But,** for many NLP applications this isn't possible because -ing is a productive suffix.
- Mean that it applies to every verb.
- Similarly -s applies to almost every noun.
- Productive suffixes even apply to new words; thus the new word fax can automatically be used in the -ing form

1. Words and Transducers (Some Concepts) (Cont..)

- Now in next section, we will survey MORPHOLOGICAL KNOWLEDGE for English language and then study some algorithms to solve these problems.

2. Survey of English Morphology

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

  - A **Morpheme** is often defined as the minimal meaning-bearing unit in a language.

For example

- the word fox consists of a single morpheme (the morpheme fox).
- while, the word cats consists of two: (i) the morpheme cat and (ii) the morpheme -s.

2. Survey of English Morphology (Cont..)

- Previous example suggests, it is often useful to distinguish two broad classes of morphemes:

  (1) **stems** and (2) **affixes.**

- The **stem** is the "main" morpheme of the word, supplying the main meaning.

  - example; In Cat's, Cat is stem.

- The **affixes** add "additional" meanings of various kinds.

  - example; In Cat's, 's is affixes.

2. Survey of English Morphology
2.1 Categories of Affixes

- **Affixes are further divided** into 4 types;
  (1) prefixes, (2) suffixes, (3) infixes, and (4) circumfixes.

(1) Prefixes precede the stem,

e.g., The word unbuckle is composed of a **stem** buckle and the prefix un-.

(2) Suffixes follow the stem,

e.g., the word eats is composed of a **stem** eat and the suffix -s.

(3) Infixes, are inserted inside the stem.

- a morpheme is inserted in the middle of a word.

e.g., the affix e, infixed to the **stem** bled "borrow" to produce bleed.

the affix um, infixed to the **stem** hingi "borrow" to produce humingi.

2. Survey of English Morphology
2.1 Categories of Affixes (Cont..)

(4) Circumfixe, circumfixes do both (prefixes and suffixes).

- English doesn't have any good examples of circumfixes, but many other languages do. In German,

e.g., adding ge- to the beginning of the stem and -t to the end;

so the past participle of the verb sagen (to say) is gesagt (said).

- Words can have more than one affix

e.g., word **"rewrites"** have

- **prefix "re"**,
- the **stem "write"** and
- **suffix "s"**

3. Morphology to create Words

- There are many ways to combine morphemes to create words.
- Four methods are common and play important roles in speech and language processing:
  (1) Inflection,
  (2) Derivation,
  (3) Cliticization, and
  (4) Compounding.

3. Morphology to create Words (Cont..)

1. Inflection

- It is the combination of a word stem with a grammatical morpheme,
- usually resulting in a word of the same class as the original stem, and usually filling some syntactic function like agreement.

- English has the inflectional morpheme **-s** for marking the plural on nouns, and
- the inflectional morpheme **-ed** for marking the past tense on verbs

For example: Play > Play**ed**

Player > Player**s**

3.1 Inflectional Morphology (**a.** Nouns)

- English has simple inflectional system with;

  (a) nouns,

  (b) verbs and

  (c) some times adjectives.

- **Nouns** have two kind of inflections:

  (i) Affix that marks plural. (e.g., cat to cats)

  (ii) Affix that marks possessive (e.g., Ali's Pen)

(i) **Affix that marks plural**

- Regular plural is spelled **-s** after most nouns,
- it is spelled **-es** after words ending in **-s** (ibis/ibises), **-z** (waltz/waltzes), **-sh** (thrush/thrushes), **-ch** (finch/finches), and sometimes **-x** (box/boxes). Nouns ending in **-y** preceded by a consonant change the **-y to -i** (butterfly/butterflies).

3.1 Inflectional Morphology (**a.** Nouns) (Cont…)

(ii) **Affix that marks possessive (Tense)**

- The possessive suffix is realized by apostrophe + -s for regular singular nouns (llama's)
- Plural nouns not ending in -s (children's)

3.1 Inflectional Morphology (**b.** Verbs)

- English verbal inflection is more complicated
- English has 3 kinds of verbs;
  - **main verbs**, {direct verb, action} (*e.g.,* eat, sleep, impeach),
  - **modal verbs** {indirect verb, week action} (*e.g.,* can, will, should), and
  - **primary verbs** {supporting verb, action} (*e.g.,* be, have, do)

- We will mostly be concerned with the main and primary verbs, because it have inflectional endings.
- Of these verbs a large class are regular, that is to say all verbs of this class have the same endings marking the same functions

3.1 Inflectional Morphology (**b.** Verbs) (Cont…)

- Regular verbs (e.g. walk) have four morphological forms, as follow:
  - **stem** walk
  - **-s form** walk**s**
  - **-ing** participle walk**ing**
  - Past form or **-ed** participle walk**ed**
- These verbs are called regular because just by knowing the stem we can predict the other forms by adding one of three predictable endings and making some regular spelling changes
- Regular verbs and forms are significant in the morphology of English first because they cover a majority of the verbs, and second because the regular class is **Productive**
- **A productive class is one that automatically includes any new words that enter the language (e.g., Fax to Faxing)**

3.1 Inflectional Morphology (**b.** Verbs) (Cont…)

- The Irregular verbs are those that have some more or less idiosyncratic forms of Irregular verb inflection
- Irregular verbs in English often have five different forms, but can have as many as eight or as few as three (e.g. *cut* or *hit*).
- Note that an irregular verb can inflect in the past form (also called the **preterite**) by changing its vowel (*eat/ate*), or its vowel and some consonants (*catch/caught*), or with no change at all (*cut/cut*).

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

3.1 Inflectional Morphology (**b.** Verbs) (Cont…)

Irregular verbs Example :

- The *-s* form is used in the "habitual present" form to distinguish the -
  - third-person singular ending (*She jogs every Tuesday*) from the other choices of person and number (*I/you/we/they jog every Tuesday*).

- In addition to noting which suffixes can be attached to which stems, we need to capture the fact that a number of regular spelling changes occur at these morpheme boundaries.

For Example, a single consonant letter is doubled before adding the *–ing* and *-ed* suffixes (*beg/begging/begged*).

3. Morphology to create Words (Cont..)

2. Derivation

- is the combination of a word stem with a grammatical morpheme,
  - mainly deal with adjective, nouns and verbs.
- Resulting in a word of a different class, often with a meaning hard to predict exactly.

For example

the verb computerize can take the derivational suffix **-ation** to produce the noun computeriz**ation**.

3.2 Derivational Morphology

Case 1: Verb/Adjective to Noun :-

➢While English inflection is relatively simple compared to other languages, derivation in English is quite complex.

➢A very common kind of derivation in English is the formation of new nouns, often from verbs or adjectives. This process is called **nominalization**.

For Example:-

the suffix *-ation* produces nouns from verbs ending often in the suffix *-ize* (*computerize* → *computerization*). Here are examples of some particularly productive English nominalizing suffixes.

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

3.2 Derivational Morphology (Cont..)

Case 2: Verb/Noun to Adjective:-

- Adjectives can also be derived from nouns and verbs. Here are examples of a few suffixes deriving adjectives from nouns or verbs.

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

- Derivation in English is more complex than inflection for a number of reasons. One is that it is generally less productive; even a nominalizing suffix like *-ation*, which can be added to almost any verb ending in *-ize*, cannot be added to absolutely every verb.

3. Morphology to create Words (Cont..)

3. Cliticization

- It is the combination of a word stem with a clitic.
- A **clitic** is a morpheme that acts syntactically like a word, but is reduced in form and attached (phonologically and sometimes orthographically) to another word

For example

English morpheme **'ve** in the word " I'**ve** " is a clitic

3.3 Cliticization Morphology

- The phonological behavior of clitics is like affixes; they tend to be short and unaccented. Their syntactic behavior is more like words, often acting as pronouns, articles, conjunctions, or verbs.
- Clitics preceding a word are called **proclitics**, (e.g., *'Tis* is it is)
  - while those following *Proclitic* are **enclitics**. (e.g., *I'm*)

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

- Note that the clitics in English are ambiguous; Thus *she's* can mean *she is* or *she has*, correctly segmenting off clitics in English is simplified by the presence of the apostrophe (') .

3. Morphology to create Words (Cont..)

4. Compounding

- It is the combination of multiple word stems together.,

For example

the noun **doghouse** is the concatenation of the morpheme **dog** with the morpheme **house**.

4. Finite-State Morphological Parsing

- Inputs from English morphologically parsed in Morphological Parse Column.

English	
Input	Morphological Parse
cats	cat +N +PL
cat	cat +N +SG
cities	city +N +Pl
geese	goose +N +Pl
goose	goose +N +Sg
goose	goose +V
gooses	goose +V +1P +Sg
merging	merge +V +PresPart
caught	catch +V +PastPart
caught	catch +V +Past

4. Finite-State Morphological Parsing (Cont…)

- The second column contains the stem of each word as well as assorted morphological **features**. These features specify additional information *Feature* about the stem.

  For Example the feature;

  +N : means that the word is a noun;

  +Sg : means it is singular,

  +Pl : means it is plural.

  +PresPart : is Present Participle (ending in "ing")

  +PastPart : is Past Participle (ending in "ed")

- Note that some of the input forms (like *caught*, *goose*, *canto*, or *vino*) will be ambiguous between different morphological parses. For now, we will consider the goal of morphological parsing merely to list all possible parses.

4. Finite-State Morphological Parsing (Cont…)

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

(1)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.).

(2) Morphotactics: the model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word. For example, the fact that the English plural morpheme follows the noun rather than preceding it is a morphotactic fact.

For Example; (e.g., In Cats, Cat is stem and "s" as plural morpheme).

(3) Orthographic rules: these **spelling rules** are used to model the changes that occur in a word, usually when two morphemes combine *For Example*; (e.g., the *y→ie* spelling rule that changes *city* + *-s* to *cities* rather than *citys*).

4.1 Building a Finite-State LEXICON (Working For Words)

- A lexicon is a repository for words.
- The simplest possible lexicon would consist of an explicit list of every word of the language

For Example;

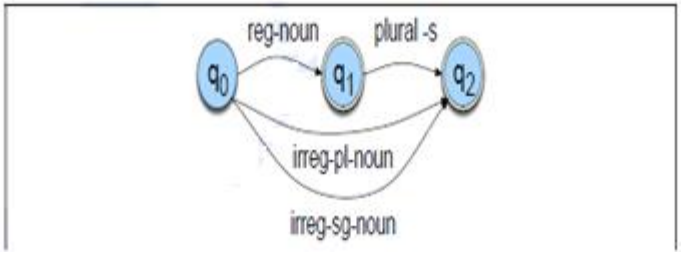
- (*every* word, i.e., including abbreviations ("AAA") and

e.g., a, AAA, AA, Aachen, aardvark, aardwolf, aba, abaca, aback, . . .

- proper names ("Jane" or "Beijing")) as follows:

- There are many ways to model morphotactics; one of the most common is the finite-state automaton.

4.2 Building a Finite-State LEXICON (Reg/Irreg Noun)



➢**Reg-noun:-** The FSA assumes that the lexicon includes regular nouns (**reg-noun**) that take the **regular -*s* plural** (e.g., *cat*, *dog*, *fox*, *aardvark*).

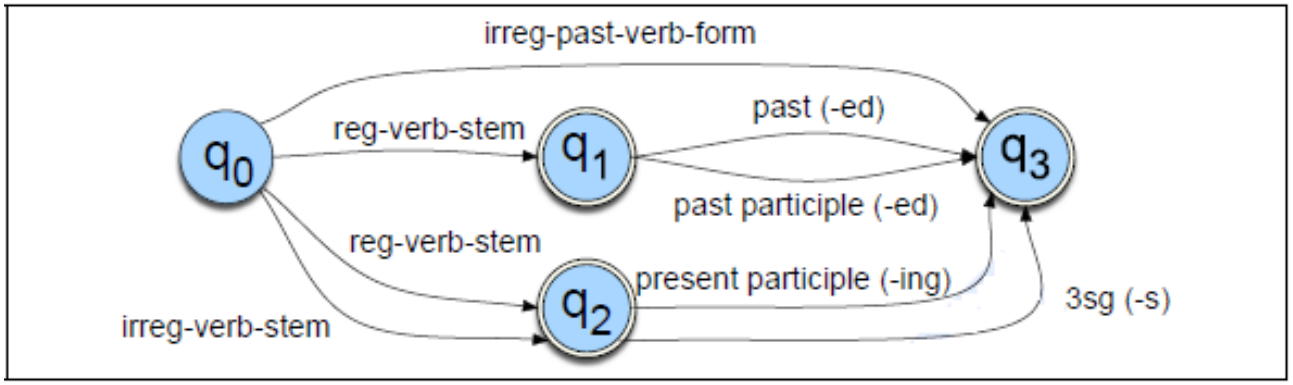
➢**irreg-pl-noun/ irreg-sg-noun :-** These are the vast majority of English nouns since for now we will ignore the fact that the plural of words like *fox* have an inserted *e*: *foxes*. The lexicon also includes irregular noun forms that don't take *-s*,

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

- both singular **irreg-sg-noun** (*goose, mouse*) and

- plural **irreg-pl-noun** (*geese,mice*).

4.3 Building a Finite-State LEXICON (Reg/ Irreg Verb)



- This lexicon has three stem classes (reg-verb-stem, irreg-verb-stem, and irreg-pastverb-form), plus four more affix classes (*-ed* past, *-ed* participle, *-ing* participle, and third singular *-s*).

Table: Lexicon for finite-state

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

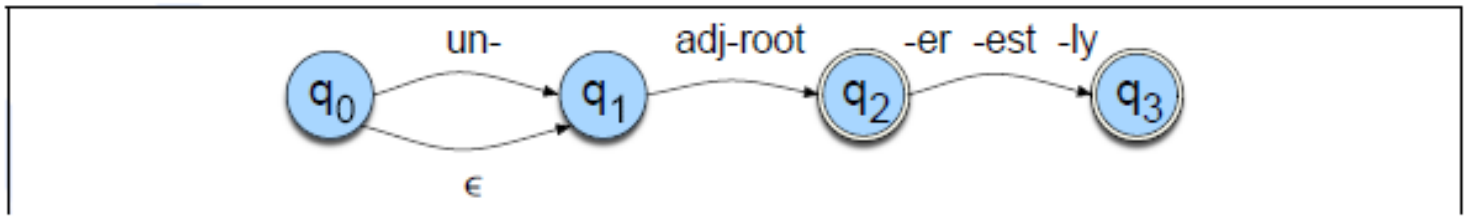
- English derivational morphology is significantly more complex than English inflectional morphology, and so automata for modeling English derivation tend to be quite complex.

4.4 Building a Finite-State LEXICON (**Example-1**)

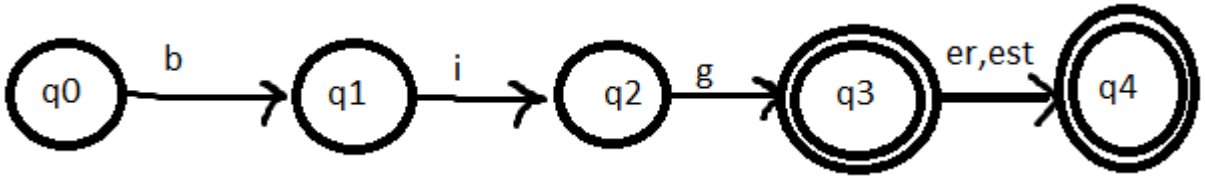
- Consider a relatively simpler case of derivation: the morphotactics of English adjectives. Here are some examples from Antworth (1990):

  e.g., big, bigger, biggest,

- 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*).



- Big word (combination);



4.4 Building a Finite-State LEXICON (**Example-1**)

Problem Defined:

- While this FSA will recognize all the adjectives, it will also recognize ungrammatical forms like *unbig*, *unfast*, *oranger*, or *smally*. We need to set up classes of roots and specify their possible suffixes.

  - Thus **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*, *small*),

- This FSA models a number of derivational facts, such as the well known generalization that any verb ending in *-ize* can be followed by the nominalizing suffix *–ation*.

CASE STUDY : -

There is a word *fossilize*, we can predict the word *fossilization* by following states *q*0, *q*1, and *q*2. Similarly, adjectives ending in *-al* or *-able* at *q*5 (*equal*, *formal*, *realizable*) can take the suffix *-ity*, or sometimes the suffix *-ness* to state *q*6 (*naturalness*, *casualness*).

4.4 Building a Finite-State LEXICON (Class Participation)

❖ Design and build a finite-state Lexicon of derivation in which morphotactics of English adjectives and FSA of following combinations are defined:

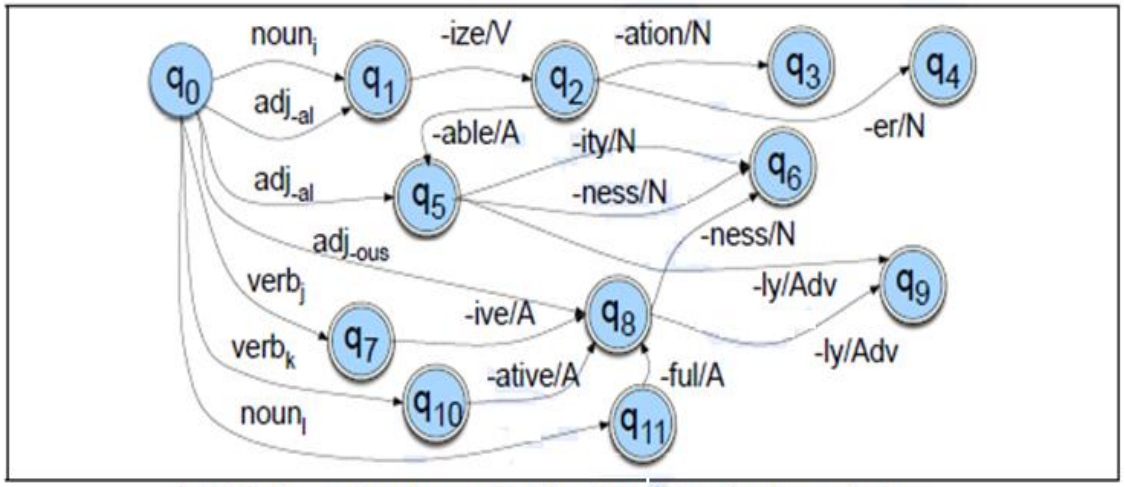
[**Note**: *design single FSA for overall word*].

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

4.4 Building a Finite-State LEXICON (Assignments)

➢ Consider the following FSA of English derivational morphology; describe following combinations of;

- ✓ q_0->q_1->q_2->q_3
- ✓ q_0->q_1->q_2->q_4
- ✓ q_0->q_5->q_6
- ✓ q_0->q_5->q_2->q_3
- ✓ q_0->q_5->q_2->q_4
- ✓ q_0->q_5->q_6
- ✓ q_0->q_5->q_9
- ✓ q_0->q_8->q_9
- ✓ q_0->q_8->q_6
- ✓ q_0->q_7->q_8->q_9
- ✓ q_0->q_{10}->q_8->q_6

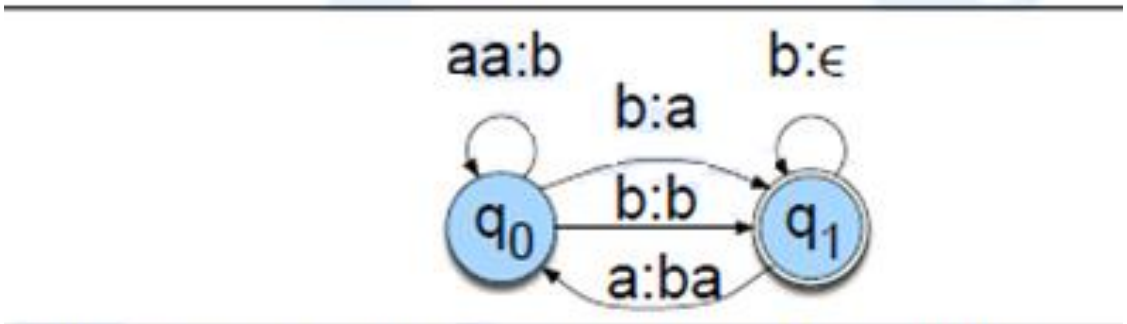


An FSA for another fragment of English derivational morphology.

- ✓ q_0->q_{10}->q_8->q_9
- ✓ q_0->q_{10}->q_8->q_6
- ✓ q_0->q_{11}->q_8->q_9
- ✓ q_0->q_{11}->q_8->q_6
- ✓ $q_0q_1q_2q_3q_4q_5q_6q_7q_8q_9q_{10}q_{11}$

5 Finite State Transducers [FST] (Working For String/ Set of Strings)

- We've now seen that FSAs can represent the morphotactic structure of a lexicon, and can be used for word recognition.
- A transducer maps between one representation and another;
- **Finite-state transducer** or **FST** is a type of finite automaton which;
  - maps between two sets of symbols. We can visualize an FST as a two-tape automaton which recognizes or generates *pairs* of strings.



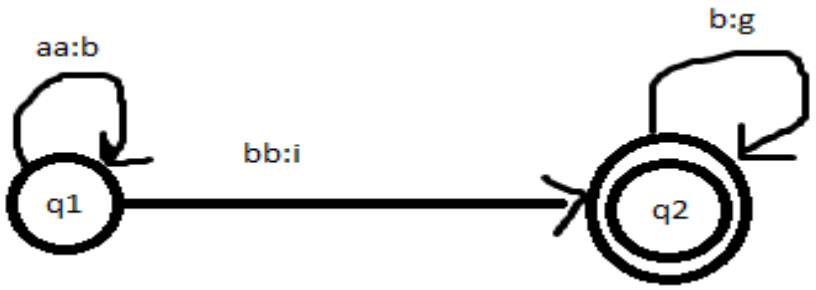
A finite-state transducer, modified from Mohri (1997).

- During FST, each arc is labeled
  - by an input and output string, separated by a colon.

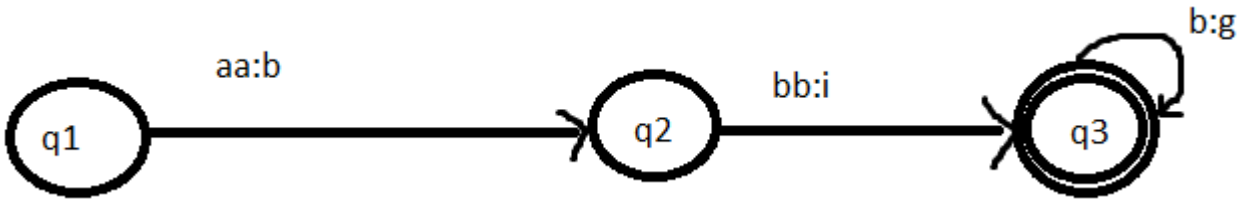
5. Finite State Transducers [FST] (Cont…)
(Working For String/ Set of Strings) [**Example**]

❖ Example; Big, bigger, biggest. [2 states; 3 states; 4 states]

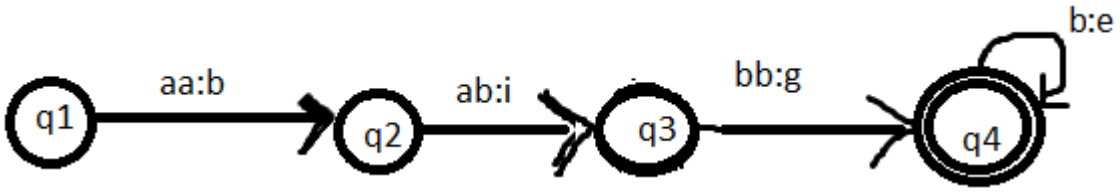
- Case 1: "2 states"



- Case 2: "3 states"



- Case 3: "4 states"



5.1 Finite State Transducers (FST)
(a. Types of FST)

- The FST 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.
- Another way of looking at an FST is as a machine that reads one string and generates another.

❖ Here are **4 ways** of thinking about transducers:

(1) 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.

(*e.g;* he go:goes to school. He goes to bazar.).

5.1 Finite State Transducers (FST)

(a. Types of FST) (Cont…)

(2) 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.

(*e.g;* She like mercedes car. His choice of car's color is red [Yes/No]).

(3) FST as translator:

- A machine that reads a string and outputs another string.

(*e.g;*).

Reads String: Hello! How are you?
Outputs : I am fine, thanks.

(4) FST as set relater:

- A machine that computes relations between sets.

(*e.g;* Older men and a boy travel in a bus. He acts as guider to them during travelling).

5.2 Finite State Transducers (FST)
(**b.** Inversion Vs Composition FST)

- For morphological parsing (and for many other NLP applications), we will apply the FST as translator metaphor, taking as input a string of letters and producing as output a string of morphemes.
- An FST can be formally defined with **7 parameters**:

Q	a finite set of N states $q_0, q_1, \ldots, q_{N-1}$
Σ	a finite set corresponding to the input alphabet
Δ	a finite set corresponding to the output alphabet
$q_0 \in Q$	the start state
$F \subseteq Q$	the set of final states
$\delta(q,w)$	the transition function or transition matrix between states; Given a state $q \in Q$ and a string $w \in \Sigma^*$, $\delta(q,w)$ returns a set of new states $Q' \in Q$. δ is thus a function from $Q \times \Sigma^*$ to 2^Q (because there are 2^Q possible subsets of Q). δ returns a set of states rather than a single state because a given input may be ambiguous in which state it maps to.
$\sigma(q,w)$	the output function giving the set of possible output strings for each state and input. Given a state $q \in Q$ and a string $w \in \Sigma^*$, $\sigma(q,w)$ gives a set of output strings, each a string $o \in \Delta^*$. σ is thus a function from $Q \times \Sigma^*$ to 2^{Δ^*}

5.2 Finite State Transducers (FST)
(b. Inversion Vs Composition FST) (Cont…)

- FSTs and regular relations are closed under union, in general they are not closed under difference, complementation and intersection.

- Besides union, FSTs have two additional closure properties;

(1) Inversion: The inversion of a transducer T (T−1) simply switches the input and output labels.

- Thus, if T maps from the input alphabet I to the output alphabet O, T−1 maps from O to I.

SYNTAX: T > Input: A – Output :Z T-1 > Input: Z – Output: A

(*e.g;* Older men (A) and a boy (Z) travel in a bus. He (Z) acts as guider to them (A) during travelling).

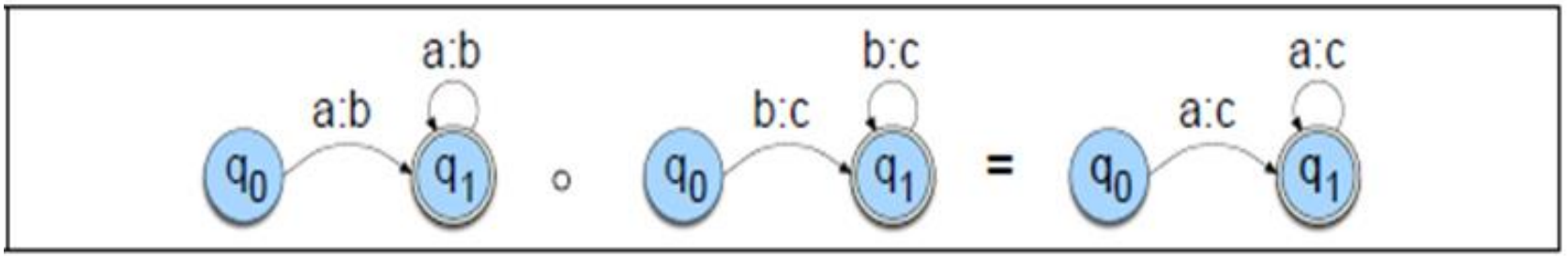
5.2 Finite State Transducers (FST)
(b. Inversion Vs Composition FST) (Cont…)

(2) Composition: If *T*1 is a transducer from *I*1 to *O*1 and *T*2 a transducer from *O*1 to *O*2, then *T*1 ∘ *T*2 maps from *I*1 to *O*2. example;

SYNTAX: T1> Input1: A – Output1: E T2> Output1: E – Output2: G

FST-based Composition

- Composition is useful because it allows us to take two transducers that run in series and replace them with one more complex transducer.
- Composition works as in algebra; applying *T*1 ∘ *T*2 to an input sequence *S* is identical to applying *T*1 to *S* and then *T*2 to the result; thus *T*1 ∘ *T*2(*S*) = *T*2(*T*1(*S*)).

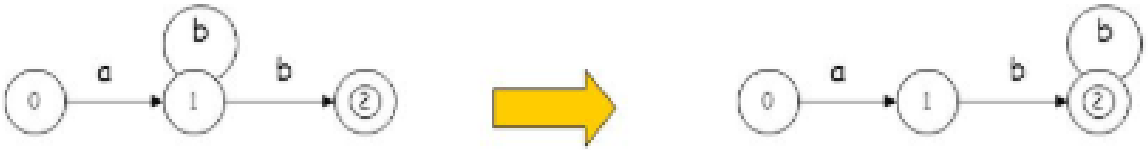


The composition of [a:b]+ with [b:c]+ to produce [a:c]+.

(*e.g;* Ali (a) and Aliya (b) are married together. Aliya (b) has two children (c)).

5.3 Finite State Transducers (FST)
(**c.** Sequential Transducers and Determinism)

- **Sequential transducers**, by contrast, are a subtype of transducers that are deterministic on their input.
- Sequential transducers are not necessarily sequential on their output.

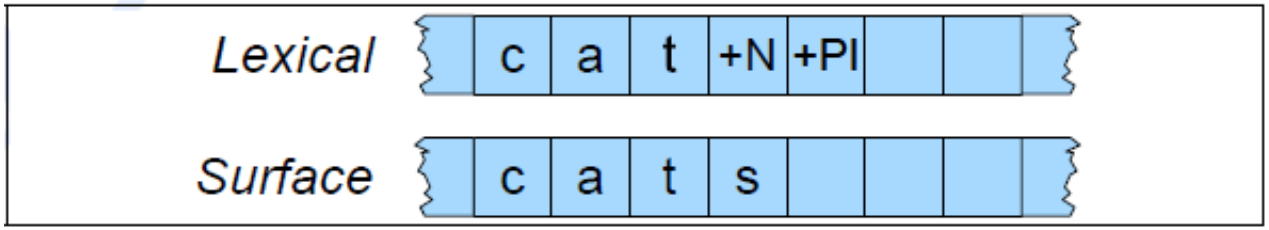


- The **sub-sequential transducer**, generates an additional output string at the final states, concatenating it onto the output produced so far.

- Generalization of sub-sequential transducers is the p-sub-sequential transducer.

6. FSTs For Morphological Parsing

- In the **finite-state morphology** paradigm, we represent a word as a correspondence between a **lexical level**, which represents a concatenation of morphemes making up a word, and
- the **surface level**, which represents the concatenation of letters which make up the actual spelling of the word.



Schematic examples of the lexical and surface tapes; the actual transducers will involve intermediate tapes as well.

- For finite-state morphology, it's convenient to view an FST as having two tapes.

  - The **upper (i.e., symbol a)** or **lexical tape**, is composed of characters from one alphabet S.

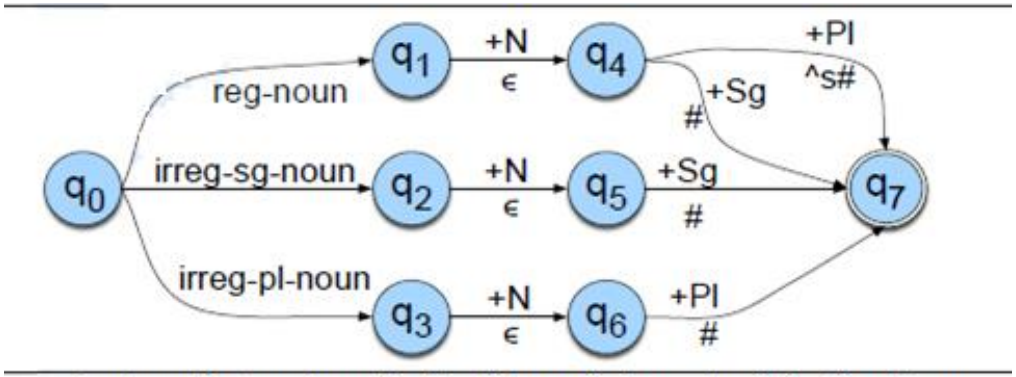
  - The **lower (i.e., symbol b)** or **surface** tape, is composed of characters from another alphabet D.

6. FSTs For Morphological Parsing (Cont…)

- In two-level morphology, the pairs of symbols in S′ are also called **feasible pairs**.
- Each feasible pair symbol *a* : *b* in the transducer alphabet S′ expresses how the symbol *a* from one tape is mapped to the symbol *b* on the other tape.
  - For example *a* : ϱ means that an *a* on the upper tape will correspond to *nothing* on the lower tape.
- The symbol ^ indicates a **morpheme replacement** (i.e., o:^e), while the symbol # indicates a **word boundary**.

6. FSTs For Morphological Parsing (Cont…)

(Example)



A schematic transducer for English nominal number inflection T_{num}.

- Transducer will map plural nouns into the stem plus the morphological marker +Pl, and singular nouns into the stem plus the morphological marker +Sg.

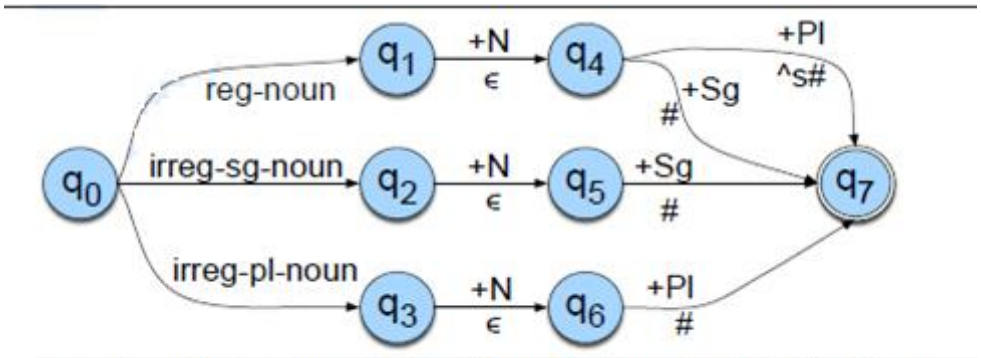
For Example;

- A surface *cats* will map to cat +N +Pl. This can be viewed in feasible-pair format as

c:c a:a t:t +N:ϙ +Pl:^s# [reg-noun] [ϙ = nothing]

p:p e:e o:o p:p l:l e:e +N:ϙ +Sg: ϙ [irreg-sg-noun]

6. FSTs For Morphological Parsing (Cont…)



A schematic transducer for English nominal number inflection T_{num}.

- In order to use a morphological noun parser, it needs to be expanded with all the individual regular and irregular noun stems, replacing the labels **reg-noun** etc.
- 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 lexical goose, the new lexical entry will be "g:g o:e o:e s:s e:e".

g:g o:e o:e s:s e:e +N:ǫ +Pl:(o:^e o:^e)# [irreg-pl-noun]

6. FSTs For Morphological Parsing (Problem definition)

- Since the output symbols include the morpheme and word boundary markers ˆ and #, the lower labels do not correspond exactly to the surface level.

- We refer to tapes with these morpheme boundary markers as **intermediate** tapes.

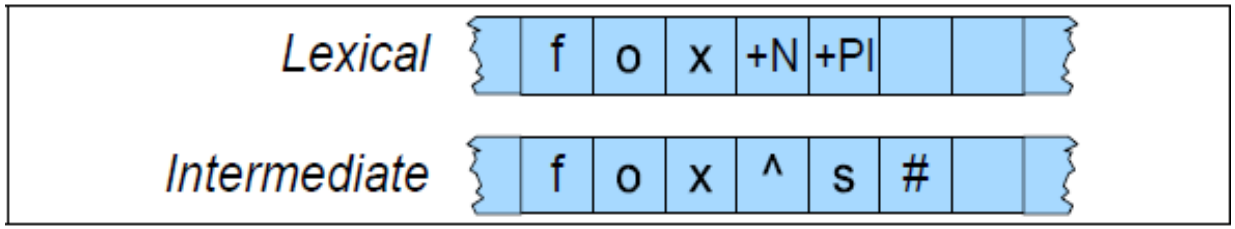


Figure 3.15 A schematic view of the lexical and intermediate tapes.

7. Transducers and Orthographic Rules

- The Previous method will successfully recognize words like *aardvarks* and *mice*.
- 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**).

Some 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

7. Transducers and Orthographic Rules (Cont…)

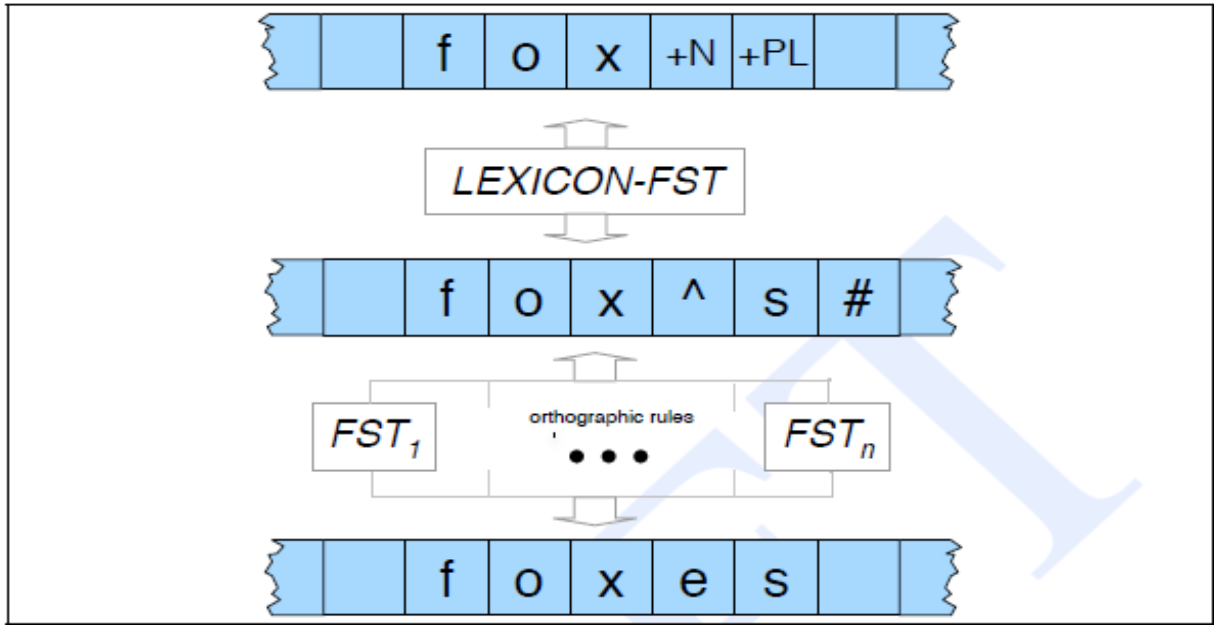
- We could write an E-insertion rule that performs the mapping from the intermediate to surface levels shown.
- Such a rule might say something like "insert an *e* on the surface tape just when the lexical tape has a morpheme ending in (*s, z, x, ch, sh* etc.) and the next morpheme is -*s*".
- Here's a formalization of the rule

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

This is the rule notation of Chomsky and Halle (1968);

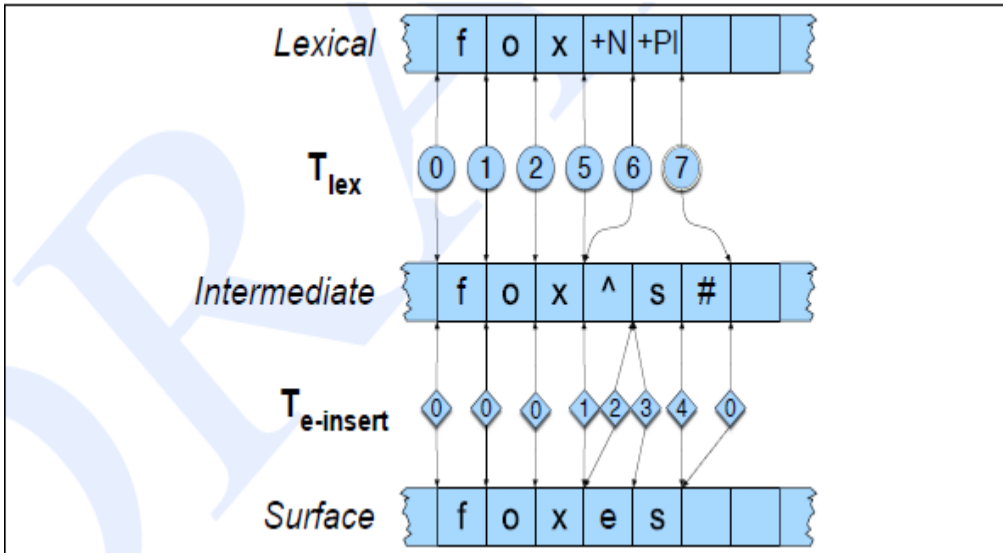
8. Combining FST Lexicon and Rules

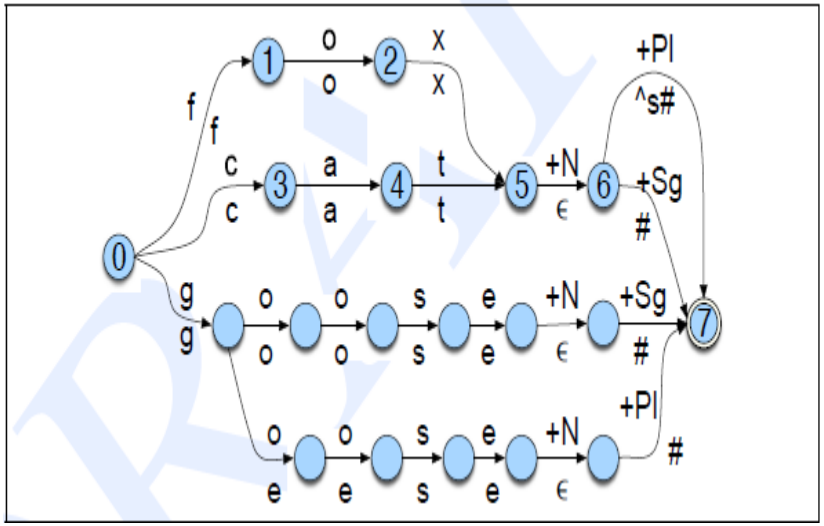
- The lexicon transducer maps between the lexical level, with its stems and morphological features, and an intermediate level that represents a simple concatenation of morphemes.
- Then a host of transducers, each representing a single spelling rule constraint, all run in parallel so as to map between this intermediate level and the surface level.



8. Combining FST Lexicon and Rules (Cont…)

- The architecture is a **two-level cascade of transducers**. Cascading two automata means running them in series with the output of the first feeding the input to the second.
- The cascade can be run top-down to generate a string, or bottom-up to parse it.





A trace of the system *accepting* the mapping from fox +N +PL to *foxes*.

8. Combining FST Lexicon and Rules (Class Participation)

❖ Design architecture of 2nd level cascade of transducers by considering combination of FST lexicon and rules :

[**Note**: *Draw Lexical* + *Intermediate* + *surface*] & [FST lexicon].

- ✓ She eats (ate/eaten) her lunch;
- ✓ He creeps (crept) the door;
- ✓ The wall shakes (shook/shaken);
- ✓ They tear (tore/torn) car;
- ✓ Animals lie (lay/lain) at road;
- ✓ Ali and azhar swim (swam/swum) smartly;

8. Combining FST Lexicon and Rules (Cont…)

- Parsing can be slightly more complicated than generation, because of the problem of **ambiguity**.
- For example, *foxes* can also be a verb and hence the lexical parse for *foxes* could be fox +V +3Sg as well as fox +N +PL.
- For ambiguous cases of this sort, the transducer is not capable of deciding. **Disambiguating** will require some external evidence such as the surrounding words.

Example

- Thus *foxes* is likely to be a noun in the sequence "*I saw two foxes yesterday*" but a verb in the sequence "*That trickster foxes me every time!*."

9. Lexicon-Free FSTs: The Porter Stemmer

Porter stemmer

The Porter stemming algorithm is a process for removing suffixes from words in English. Removing suffixes automatically is an operation which is especially useful in the field of information retrieval. In a typical IR environment a document is represented by a vector of words, or terms. Terms with a common stem will usually have similar meanings, for example:

```
CONNECT
CONNECTED
CONNECTING
CONNECTION
CONNECTIONS
```

Frequently, the performance of an IR system will be improved if term groups such as this are conflated into a single term. This may be done by removal of the various suffixes -ED, -ING, -ION, IONS to leave the single term CONNECT. In addition, the suffix stripping process will reduce the total number of terms in the IR system, and hence reduce the size and complexity of the data in the system, which is always advantageous.

9. Lexicon-Free FSTs: The Porter Stemmer (Cont…)

- the Porter algorithm also can be viewed as a lexicon-free FST stemmer. The algorithm contains a series of rules like these

  ATIONAL → ATE (e.g., relational→relate)

  ING → ϱ if stem contains vowel (e.g., motoring→ motor)

  SSES → SS (e.g., grasses→ grass)

- Stemming tends to improve the performance of information retrieval (IR), especially with smaller documents.

  e.g., overwrite or replace function in MS word.

10. Word and Sentence Tokenization

- Word tokenization may seem very simple in a language like English that separates words via a special 'space' character.
- A closer examination will make it clear that whitespace is not sufficient by itself.

For Example;

Consider the following sentences from a Wall Street Journal and New York Times article, respective

Sentence 1(Wall Street Journal)

Mr. Sherwood said reaction to Sea Containers' proposal has been "very positive." In New York Stock Exchange composite trading yesterday, Sea Containers closed at $62.625, up 62.5 cents.

Sentence 2(New York Times article)

"I said, 'what're you? Crazy?' "said Sadowsky. "I can't afford to do that."

10. Word and Sentence Tokenization (Cont…)

- Segmenting purely on white-space would produce words like these:

  cents. *said,* *positive."* *Crazy?*

- We could address these errors by treating punctuation, in addition to whitespace, as a word boundary.

Problems of word tokenization:

- Punctuation often occurs word internally.

  Example: *m.p.h,*, *Ph.D.*, *AT&T*, *cap'n*, *01/02/06*, and *google.com*.

- Similarly, assuming that we want *62.5* to be a word, we'll need to avoid segmenting every period, since that will segment this into 62 and 5.
- Another useful task a tokenizer can do for us is to expand clitic contractions that are marked by apostrophes,

  for example converting *what're* to the two tokens *what are*, and

  we're to *we are*.

10. Word and Sentence Tokenization (Cont…)

- Tokenization algorithms may also tokenize multiword expressions like *New York* or *rock 'n' roll*, which requires a multiword expression dictionary of some sort.
- This makes tokenization intimately tied up with the task of detecting names, dates, and organizations, which is called *named entity detection*.
- In addition to word segmentation, **sentence segmentation** is a crucial first step in text processing.
- **Segmenting** a text into sentences is generally based on punctuation. This is because certain kinds of punctuation (periods, question marks, exclamation points) tend to mark sentence boundaries.
- Question marks and exclamation points are relatively unambiguous markers of sentence boundaries.

Problems of sentence tokenization:

- The period character '.' is ambiguous between a sentence boundary marker and a marker of abbreviations like *Mr.* or *Inc.*

10. Word and Sentence Tokenization

(Presentation of each candidate)

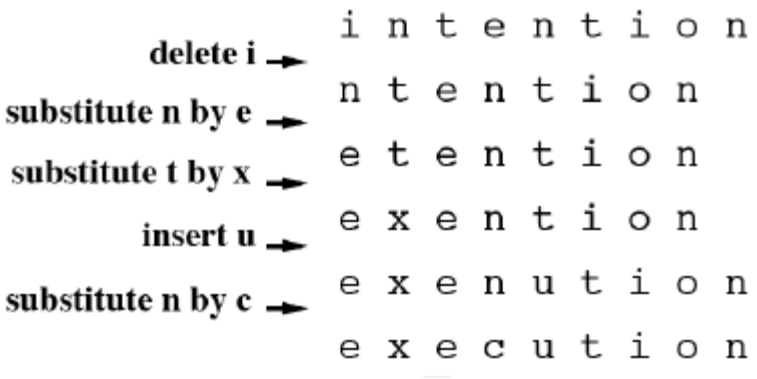
Solutions of Word/sentence tokenization:

11. Minimum Edit Distance

- The distance between *String distance* two strings is a measure of how alike two strings are to each other.
- The minimum edit distance between two strings is the minimum number of editing operations (insertion, deletion, substitution) needed to transform one string into another.
- For example the gap between the words *intention* and *execution* is five operations

11. Minimum Edit Distance (Cont…)

- The minimum edit distance is computed by **dynamic programming**. Dynamic programming is the name for a class of algorithms, that apply a table-driven method to solve problems by combining solutions to subproblems.
- This class of algorithms includes the most commonly-used algorithms in speech and language processing.
- The intuition of a dynamic programming problem is that a large problem can be solved by properly combining the solutions to various subproblems.
- For example, consider the sequence or "path" of transformed words that comprise the minimum edit distance between the strings *intention* and *execution*



11. Minimum Edit Distance (Cont…)

- Dynamic programming algorithms for sequence comparison work by creating a distance matrix with one column for each symbol in the target sequence and one row for each symbol in the source sequence (i.e., target along the bottom, source along the side).
- For minimum edit distance, this matrix is the *edit-distance* matrix. Each cell *edit-distance*[*i*,*j*] contains the distance between the first *i* characters of the target and the first *j* characters of the source.
- Each cell can be computed as a simple function of the surrounding cells; thus starting from the beginning of the matrix it is possible to fill every entry.
- The value in each cell is computed by taking the minimum of the three possible paths through the matrix which arrive there.

$$distance[i,j] = \min \begin{cases} distance[i-1,j] + \text{ins-cost}(target_{i-1}) \\ distance[i-1,j-1] + \text{sub-cost}(source_{j-1}, target_{i-1}) \\ distance[i,j-1] + \text{del-cost}(source_{j-1}) \end{cases}$$