COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

Modeling language as a sequence of tokens

CMSC 470

Marine Carpuat

Beyond MT: Encoder-Decoder can be used as Conditioned Language Models P(Y|X) to generate text Y based on some input X

Input *X*	Output *Y* (**Text**)	Task
Structured Data	NL Description	NL Generation
English	Japanese	Translation
Document	Short Description	Summarization
Utterance	Response	Response Generation
Image	Text	Image Captioning
Speech	Transcript	Speech Recognition

Given some text, how to segment it into a sequence of tokens?

Turn this text into a sequence of tokens

They’re not family or close friends, and they often don’t know Makris by name.

https://dbknews.com/2019/10/23/high-five-guy-umd-checking-in-umd-legend/

Turn this text into a sequence of tokens

姚明进入总决赛

Example from Martin & Jurafsky chap 2

Turn this text into a sequence of tokens

uygarlaştıramadıklarımızdanmışsınızcasına

(Meaning: *behaving as if you are among those whom we could not cause to become civilized)*

Basic preprocessing steps to get a sequence of tokens from running text

- Sentence segmentation: break up a text into sentences
  - Based on cues like periods or exclamation points

- Tokenization: task of separating out words in running text
  - Can be handled by rules/regular expressions
  - Split on whitespace is often not sufficient
  - Additional rules needed to handle punctuation, abbreviations, emoticons, hashtags…

- Normalization to minimize sparsity:
  - Normalize case, punctuation, encoding of diacritics in Unicode…

Vocabulary issues with neural sequence-to-sequence models

- Out of vocabulary words
  - the neural encoder-decoder models we've seen have a closed vocabulary
  - how can they process/generate new words at test time?
- The larger the vocabulary, the larger the models
  - One embedding vector per word type
  - Dimension of output softmax vector increases with vocab size
- How can we reduce the model's vocabulary size without restricting the nature of language it can model?

Can we model text as sequences of characters instead of sequences of words?

Character level models

- View text as sequence of characters rather than sequences of words
- Pro: Character vocabulary is smaller than word vocabulary
- Con: Sequences are longer

If naively implemented as an RNN

- RNN composition function should capture both how words are formed and how sentences are formed
- Character embeddings perhaps not as useful as word embeddings

Open research question: can we design neural architectures that model words and characters jointly?

See [Ling et al. 2015; Jaech et al. 2016; Chen et al 2018, …]

Today: can we use sequences of subwords as a middle ground between word and character models?

Segmenting words into subword using Linguistic Knowledge

Morphological Analysis

Morphology

- Study of how words are constructed from smaller units of meaning
- Smallest unit of meaning = morpheme
  - fox has morpheme fox
  - cats has two morphemes cat and –s

- Two classes of morphemes:
  - Stems: supply the “main” meaning
    - Aka root / lemma
  - Affixes: add “additional” meaning

Topology of Morphologies

- Concatenative vs. non-concatenative
- Derivational vs. inflectional
- Regular vs. irregular

Concatenative Morphology

- Morpheme+Morpheme+Morpheme+…
- Stems (also called lemma, base form, root, lexeme):
  - hope+ing → hoping
  - hop+ing → hopping
- Affixes:
  - Prefixes: Antidis**establish**mentarianism
  - Suffixes: Antidis**establish**mentarianism
- Agglutinative languages (e.g., Turkish)
  - uygarlaştıramadıklarımızdanmışsınızcasına → uygar+laş+tır+ama+dık+lar+ımız+dan+mış+sınız+casına
  - Meaning: *behaving as if you are among those whom we could not cause to become civilized*

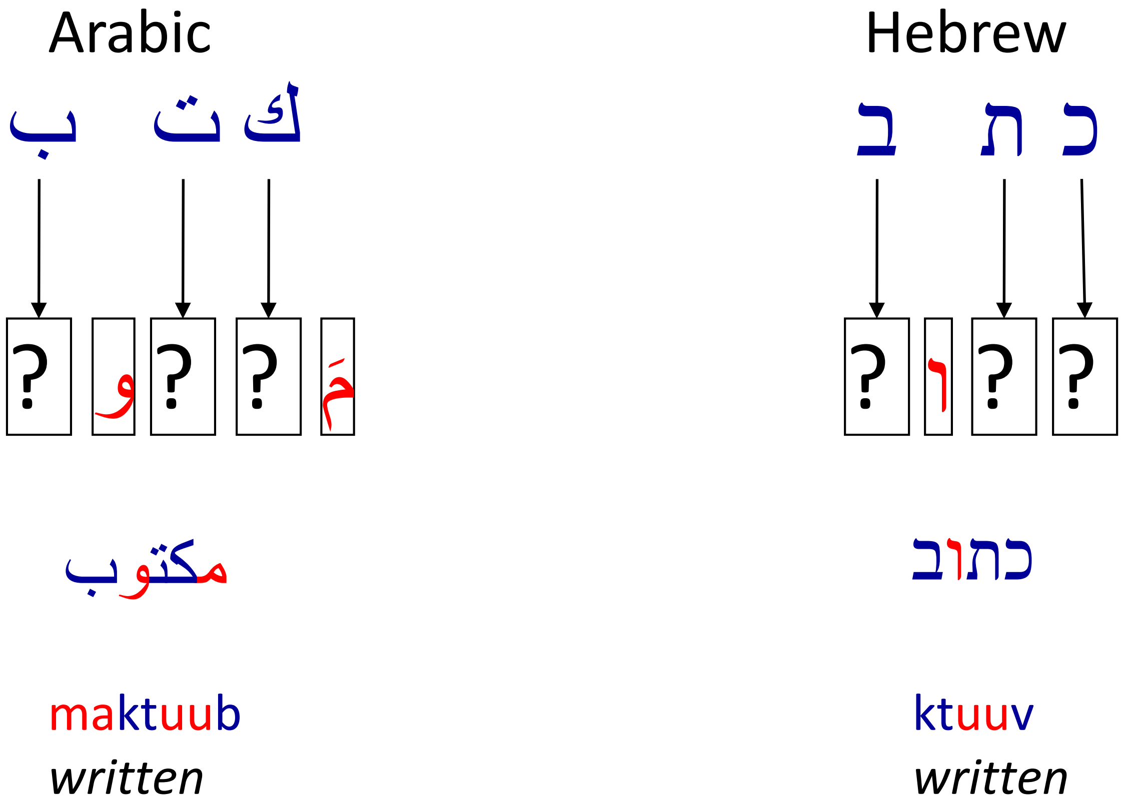
Non-Concatenative Morphology

- Infixes (e.g., Tagalog)
  - hingi (borrow)
  - humingi (borrower)

- Circumfixes (e.g., German)
  - sagen (say)
  - gesagt (said)

Templatic Morphologies

- Common in Semitic languages
- Roots and patterns



Inflectional Morphology

- Stem + morpheme →
  - Word with same part of speech as the stem
- Adds: tense, number, person,…

- Plural morpheme for English noun
  - cat+s
  - dog+s
- Progressive form in English verbs
  - walk+ing
  - rain+ing

Derivational Morphology

- Stem + morpheme →
  - New word with different meaning or different part of speech
  - Exact meaning difficult to predict

- Nominalization in English:
  - -ation: computerization, characterization
  - -ee: appointee, advisee
  - -er: killer, helper

- Adjective formation in English:
  - -al: computational, derivational
  - -less: clueless, helpless
  - -able: teachable, computable

Noun Inflections in English

- Regular
  - cat/cats
  - dog/dogs
- Irregular
  - mouse/mice
  - ox/oxen
  - goose/geese

Verb Inflections in English

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

Morphological Parsing

- Computationally decompose input forms into component morphemes
- Components needed:
  - A lexicon (stems and affixes)
  - A model of how stems and affixes combine
  - Orthographic rules

Morphological Parsing: Examples

WORD	**STEM (+FEATURES)**
cats	cat +N +PL
cat	cat +N +SG
cities	city +N +PL
geese	goose +N +PL
ducks	(duck +N +PL) or (duck +V +3SG)
merging	merge +V +PRES-PART
caught	(catch +V +PAST-PART) or (catch +V +PAST)

Different Approaches

- Lexicon only
- Rules only
- Lexicon and rules
    - finite-state transducers

Lexicon-only

- Simply enumerate all surface forms and analyses

```
acclaim          acclaim $N$
acclaim          acclaim $V+0$
acclaimed        acclaim $V+ed$
acclaimed        acclaim $V+en$
acclaiming       acclaim $V+ing$
acclaims         acclaim $N+s$
acclaims         acclaim $V+s$
acclamation      acclamation $N$
acclamations     acclamation $N+s$
acclimate        acclimate   $V+0$
acclimated       acclimate   $V+ed$
acclimated       acclimate   $V+en$
acclimates       acclimate   $V+s$
acclimating      acclimate   $V+ing$
```

Rule-only

- Cascading set of rules
  - s → ε
  - ation → e
  - ize → ε
  - …

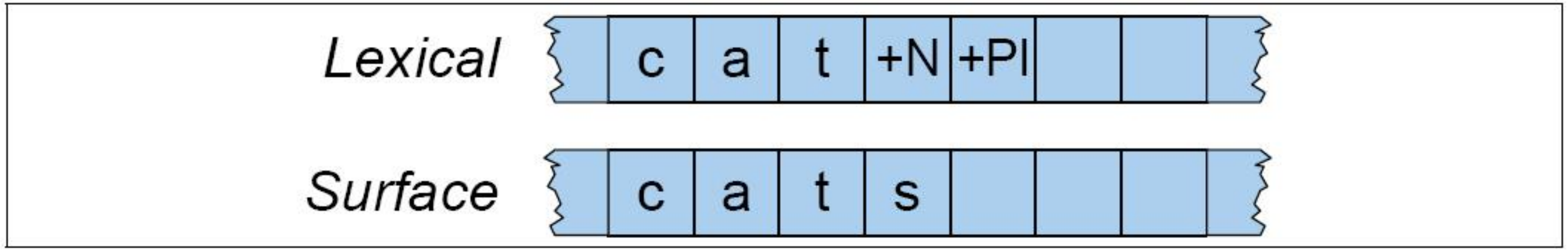
- Example
  - generalizations
    → generalization
    → generalize
    → general
  - organizations
    → organization
    → organize
    → organ

Morphological Parsing with Finite State Transducers

Combination of lexicon + rules

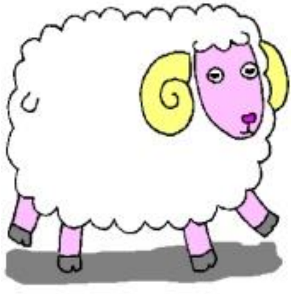
A machine that reads and writes on two tapes:

One tape contains the input, the other tape as the analysis



Finite State Automaton (FSA)

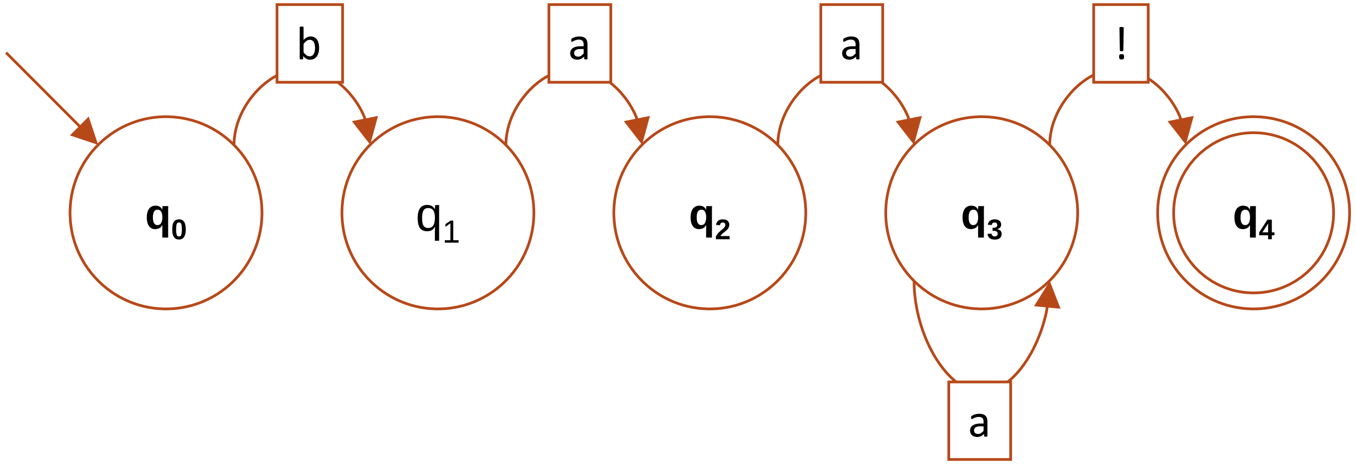
Language:



baa!
baaa!
baaaa!
baaaaa!
...

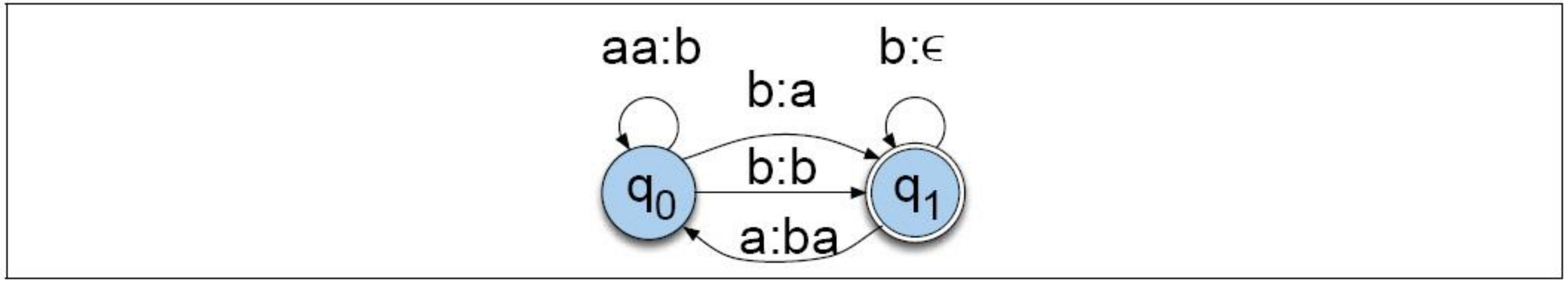
Regular Expression:
/baa+!/

Finite-State Automaton:



Finite-State Transducers (FSTs)

- A two-tape automaton that recognizes or generates pairs of strings
- Think of an FST as an FSA with two symbol strings on each arc
  - One symbol string from each tape

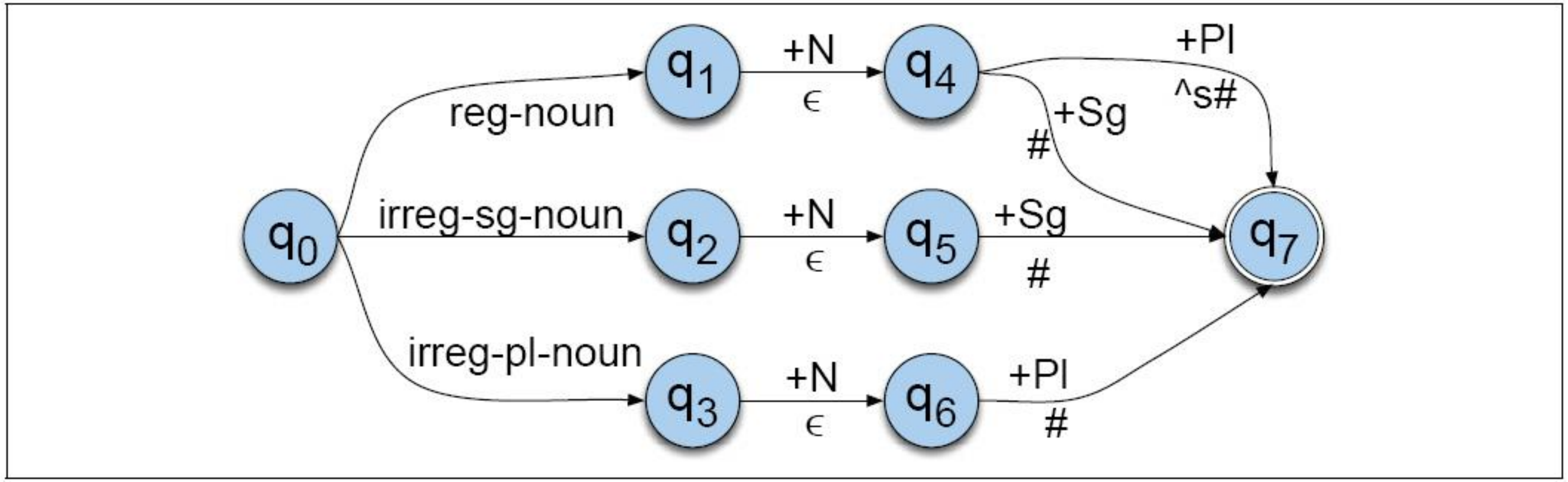


Terminology

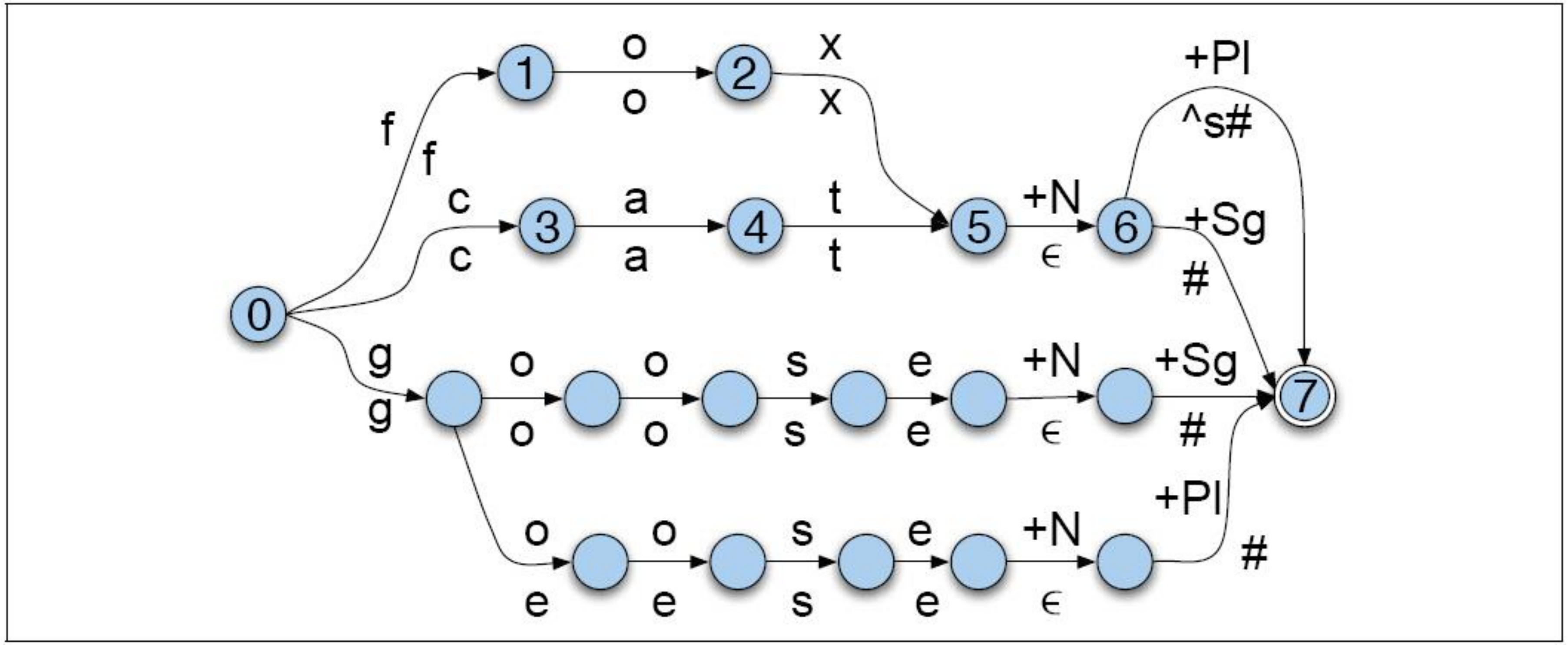
- Transducer alphabet (pairs of symbols):
  - a:b = *a* on the upper tape, *b* on the lower tape
  - a:ε = *a* on the upper tape, nothing on the lower tape
  - If a:a, write a for shorthand
- Special symbols
  - # = word boundary
  - ^ = morpheme boundary
  - (For now, think of these as mapping to ε)

FST for English Nouns

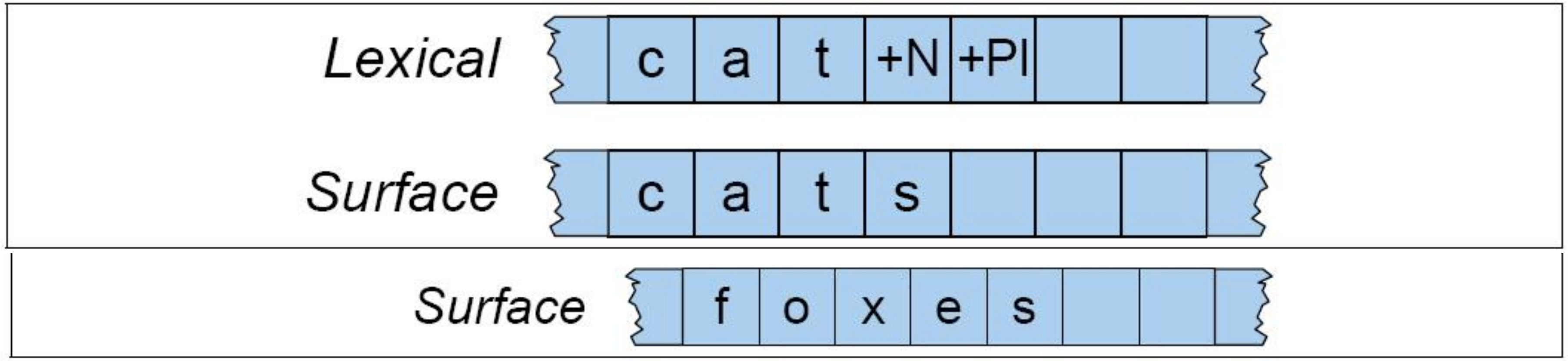
- First try:



FST for English Nouns

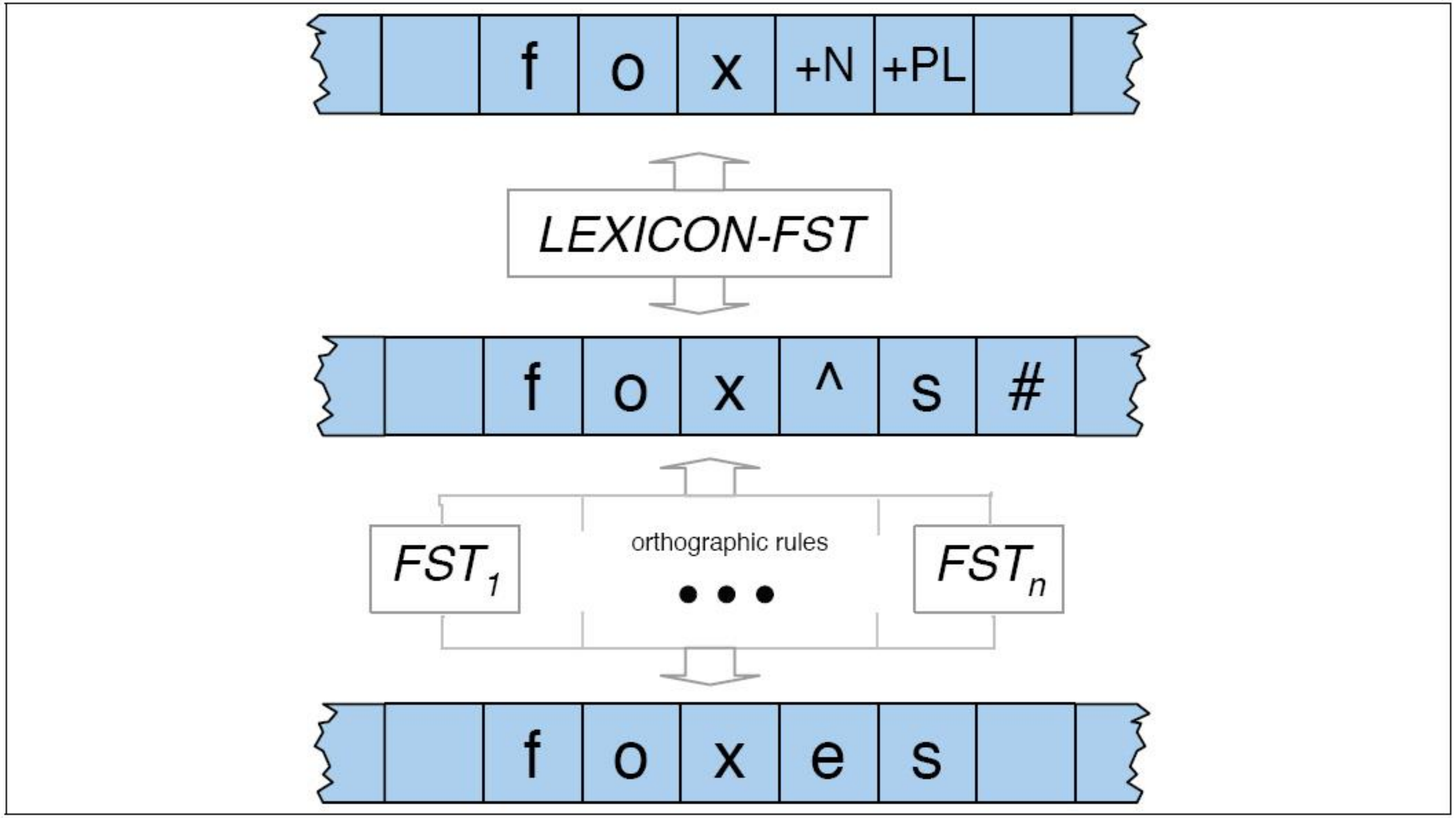


Handling Orthography



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

Complete Morphological Parser



Practical NLP Applications

- In practice, it is almost never necessary to write FSTs by hand…
- Typically, one writes rules:
  - Chomsky and Halle Notation: a → b / c__d
    = rewrite a as b when occurs between c and d
  - E-Insertion rule

$$\varepsilon \rightarrow e \,/\, \left\{ \begin{array}{c} x \\ s \\ z \end{array} \right\} \hat{}\ \_\_\ s\ \#$$

- Rule → FST compiler handles the rest…

Segmenting words into subword using counts

Byte Pair Encodings

One approach to unsupervised subword segmentation

- Goal: a kind of tokenization where
  - most tokens are words
  - but some tokens are frequent morphemes or other subwords
  - So that unseen words can be represented by combining seen subword units

- "Byte-pair encoding" (BPE) [Sennrich et al. 2016] is one technique to generate such tokenization
  - Based on a method for text compression
  - Intuition: merge frequent pairs of characters

Learning a set of subwords with the Byte Pair Encoding Algorithm

- Start state:
  - Given set of symbols = set of characters
  - Each word is represented as a sequence of character + end of word symbol "_"

- At each step:
  - Count number of symbol pairs
  - Find the most frequent pair
  - Replace it with a new merged symbol

- Terminate
  - After k merges; k is a hyperparameter
  - The resulting symbol set will consist of original characters + k new symbols

Byte Pair Encoding Illustrated

- Starting state

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

- After the first merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_
2	l o w e s t _	
6	n e w e r_	
3	w i d e r_	
2	n e w _	

Byte Pair Encoding Illustrated

- After the 2nd merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

- After the 3rd merge

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew
2	l o w e s t _	
6	n ew er_	
3	w i d er_	
2	n ew _	

Byte Pair Encoding Illustrated

- If we continue, the next merges are

Merge	**Current Vocabulary**
(n, ew)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new
(l, o’	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_, low_

Byte Pair Encoding at test time

- On a new test sentence
  - Segment each test sentence into characters and apply end of word token
  - Greedily apply merge rules in the order we learned them at training time

- E.g., given the learned subwords

$$\_, \mathrm{d}, \mathrm{e}, \mathrm{i}, \mathrm{l}, \mathrm{n}, \mathrm{o}, \mathrm{r}, \mathrm{s}, \mathrm{t}, \mathrm{w}, \mathrm{r}\_, \mathrm{er}\_, \mathrm{ew}, \mathrm{new}, \mathrm{lo}, \mathrm{low}, \mathrm{newer}\_, \mathrm{low}\_$$

- What is the BPE tokenization of
  - “newer_”?
  - “lower_”?

```python
import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i],symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape('␣'.join(pair))
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
    for word in v_in:
        w_out = p.sub(''.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

vocab = {'l␣o␣w␣</w>' : 5, 'l␣o␣w␣e␣s␣t␣</w>' : 2,
    'n␣e␣w␣e␣r␣</w>':6, 'w␣i␣d␣e␣r␣</w>':3, 'n␣e␣w␣</w>':2}
num_merges = 8

for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)
```

Figure 2.12 Python code for BPE learning algorithm from Sennrich et al. (2016).

Alternatives to BPE

- Wordpiece [Wu et al. 2016]
  - Start with some simple tokenization just like BPE
  - Puts a special word boundary token at the beginning rather than end of word
  - Merge pairs to minimize the language model likelihood of the training data

- SentencePiece [Kudo & Richardson 2018]
  - Works from raw text (no need for initial tokenization, whitespace handled like any other symbol)

Modeling language as a sequence of tokens Summary

- Segmenting running text into tokens is not a trivial task
  - White-space and punctuation-based rules provide a first cut for many languages, but are not sufficient

- The nature of the segmentation defines the size/nature of the model vocabulary
  - And whether unknown words can be processed at test time

- 2 approaches to segment words into subwords
  - Use linguistic knowledge to perform morphological analysis: segment words into morphemes
  - Using training data frequencies only: e.g., Byte-Pair Encoding algorithm