Natural Language Processing

Spring 2017

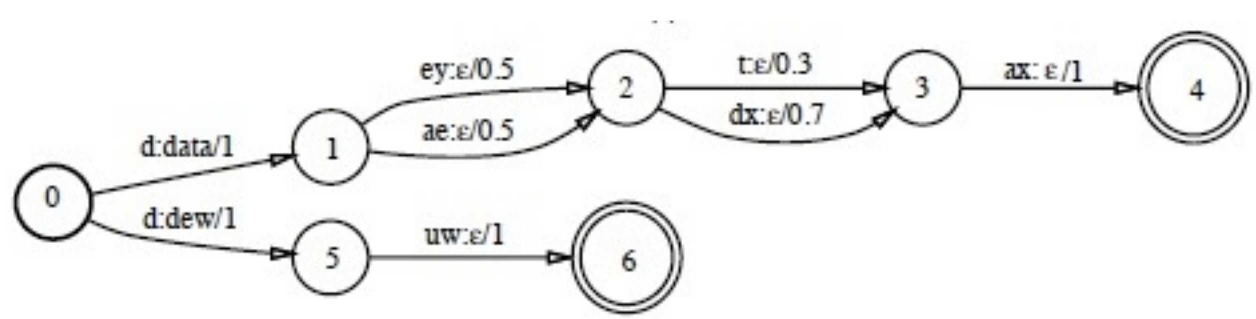
Unit 1: Sequence Models

Lectures 7-8: Stochastic String Transformations

(a.k.a. “channel-models”)

required

optional



Professor Liang Huang

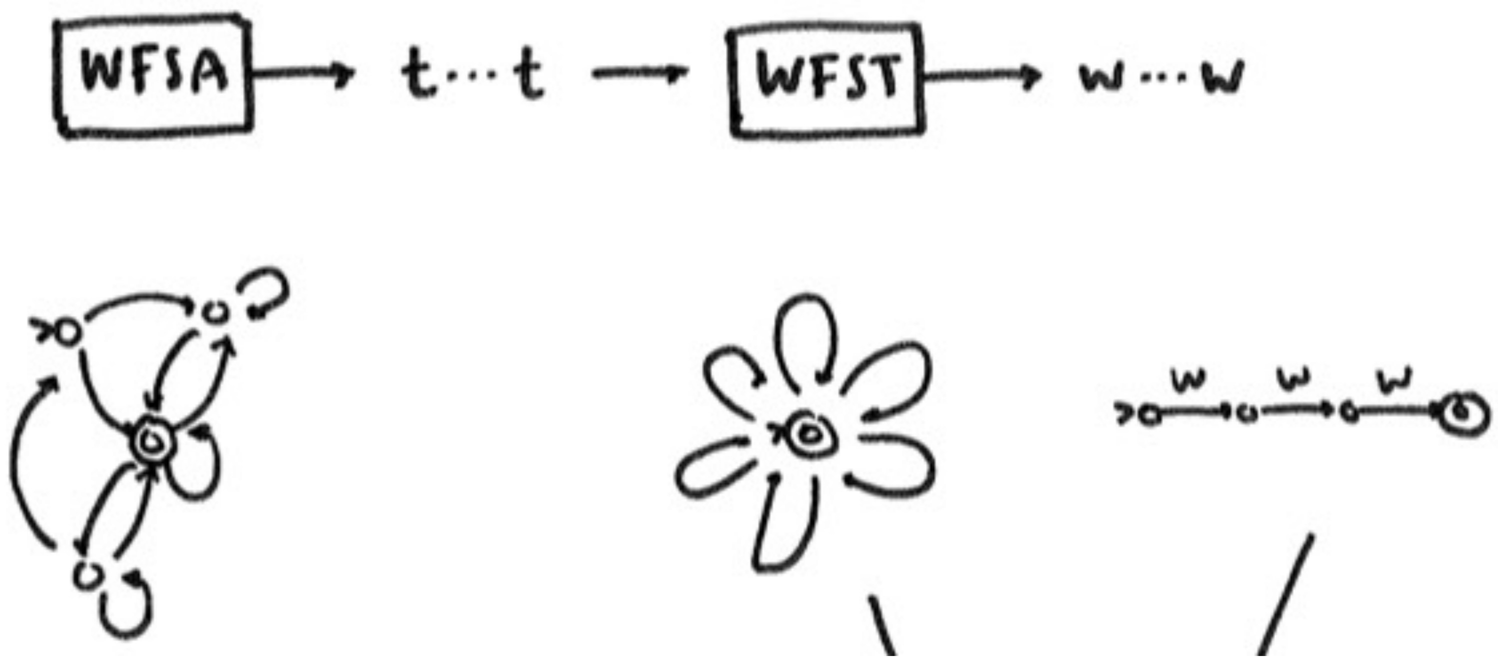
liang.huang.sh@gmail.com

String Transformations

- General Framework for many NLP problems
- Examples

  - Part-of-Speech Tagging
  - Spelling Correction (Edit Distance)
  - Word Segmentation
  - Transliteration, Sound/Spelling Conversion, Morphology
  - Chunking (Shallow Parsing)
  - Beyond Finite-State Models (i.e., tree transformations)
    - Summarization, Translation, Parsing, Information Retrieval, ...
- Algorithms: Viterbi (both max and sum)

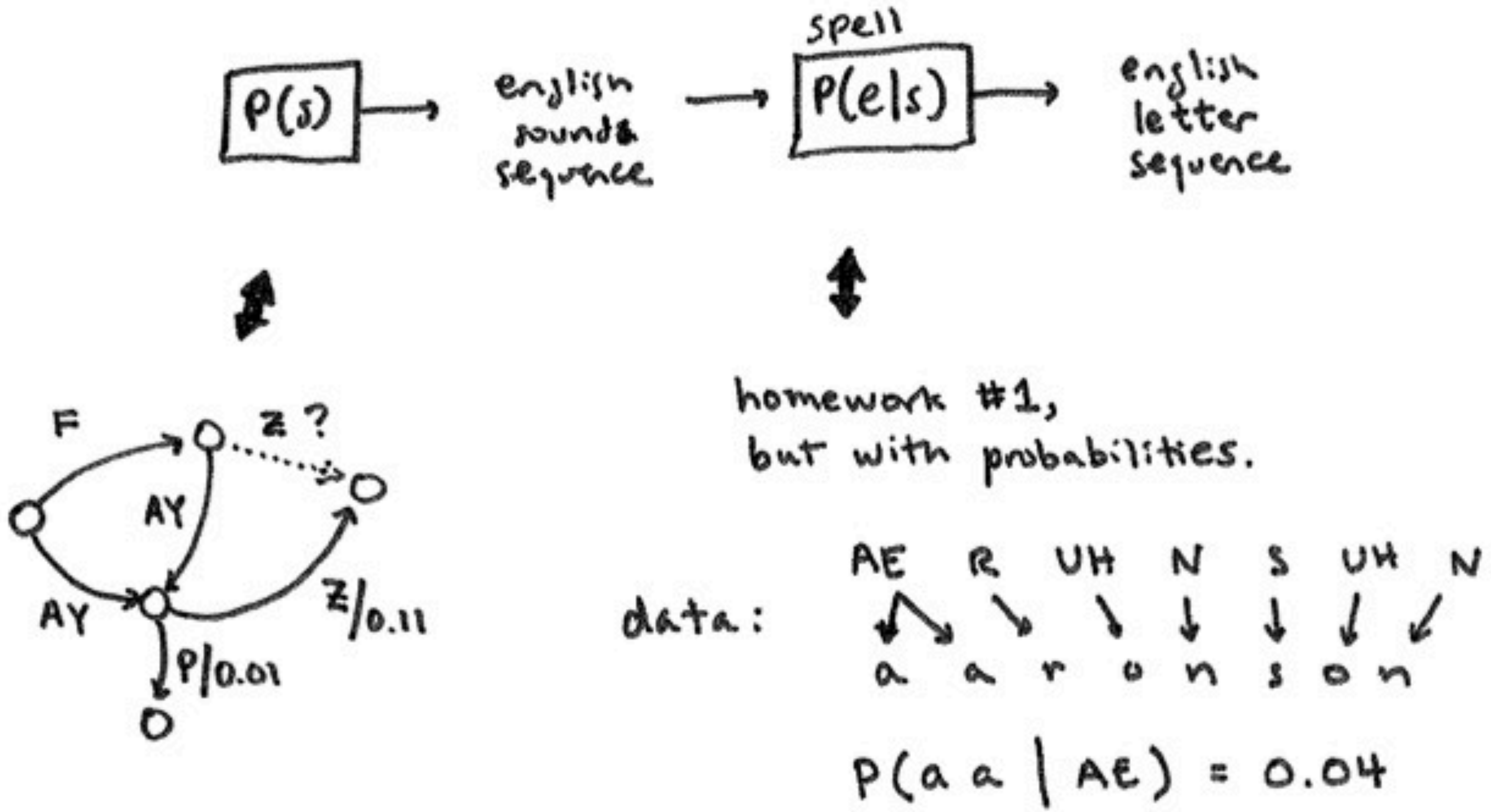
Review of Noisy-Channel Model



Application	Input	Output	p(i)	p(o|i)
Machine Translation	L_1 word sequences	L_2 word sequences	p(L_1) in a language model	translation model
Optical Character Recognition (OCR)	actual text	text with mistakes	prob of language text	model of OCR errors
Part Of Speech (POS) tagging	POS tag sequences	English words	prob of POS sequences	p(w|t)
Speech recognition	word sequences	speech signal	prob of word sequences	acoustic model

(hw2) From Spelling to Sound

- word-based or char-based



Pronunciation Dictionary

http://www.speech.cs.cmu.edu/cgi-bin/cmudict
from CMU Pronunciation Dictionary
39 phonemes (15 vowels + 24 consonants)

- (hw3: eword-epron.data)
  - ...
  - AARON EH R AH N
  - AARONSON AA R AH N S AH N
  - ...
  - PEOPLE P IY P AH L
  - VIDEO V IH D IY OW

```
echo 'W H A L E B O N E S' |
    carmel -sriIEQk 5 epron.wfsa epron-espell.wfst
```

- you can train p(s..s|w) from this, but what about unseen words?
- also need alignment to train the channel model p(s|e) & p(e|s)

CMU Dict: 39 Ame. Eng. Phonemes

WRONG! missing the SCHWA ə (merged with the STRUT ʌ "AH")!

CMU/IPA	Example	Translation
AA /ɑ/	odd	AA D
AE /æ/	at	AE T
AH /ʌ/	hut	HH AH T
AO /ɔ:/	ought	AO T
AW /aʊ/	cow	K AW
AY /aɪ/	hide	HH AY D
B /b/	be	B IY
CH /tʃ/	cheese	CH IY Z
D /d/	dee	D IY
DH /ð/	thee	DH IY
EH /ɛ/	Ed	EH D
ER /ɚ/	hurt	HH ER T
EY /eɪ/	ate	EY T
F /f/	fee	F IY
G /g/	green	G R IY N
HH /h/	he	HH IY
IH /ɪ/	it	IH T
IY /i:/	eat	IY T
JH /dʒ/	gee	JH IY

CMU/IPA	Example	Translation
K /k/	key	K IY
L /l/	lee	L IY
M /m/	me	M IY
N /n/	knee	N IY
NG /ŋ/	ping	P IH NG
OW /oʊ/	oat	OW T
OY /ɔɪ/	toy	T OY
P /p/	pee	P IY
R /ɹ/	read	R IY D
S /s/	sea	S IY
SH /ʃ/	she	SH IY
T /t/	tea	T IY
TH /θ/	theta	TH EY T AH
UH /ʊ/	hood	HH UH D
UW /u/	too	T UW
V /v/	vee	V IY
W /w/	we	W IY
Y /j/	yield	Y IY L D
Z /z/	zee	Z IY
ZH /ʒ/	usual	Y UW ZH UW AH L

CMU Pronunciation Dictionary

WRONG! missing the SCHWA ə (merged with the STRUT ʌ "AH")!
DOES NOT ANNOTATE STRESSES

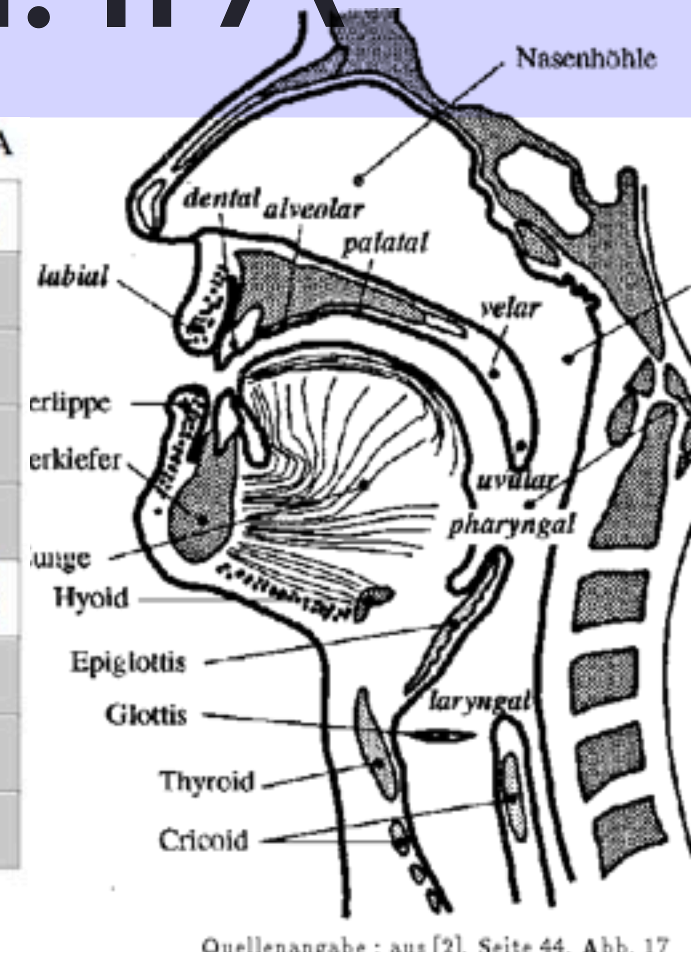
```
A           AH
A           EY
AAA         T R IH P AH L EY
AABERG      AA B ER G
AACHEN      AA K AH N
...
ABOUT       AH B AW T
...
ABRAMOVITZ  AH B R AA M AH V IH T S
ABRAMOWICZ  AH B R AA M AH V IH CH
ABRAMOWITZ  AH B R AA M AH W IH T S
...
FATHER      F AA DH ER
...
ZYDECO      Z AY D EH K OW
ZYDECO      Z IH D AH K OW
ZYDECO      Z AY D AH K OW
...
ZZZZ        Z IY Z
```

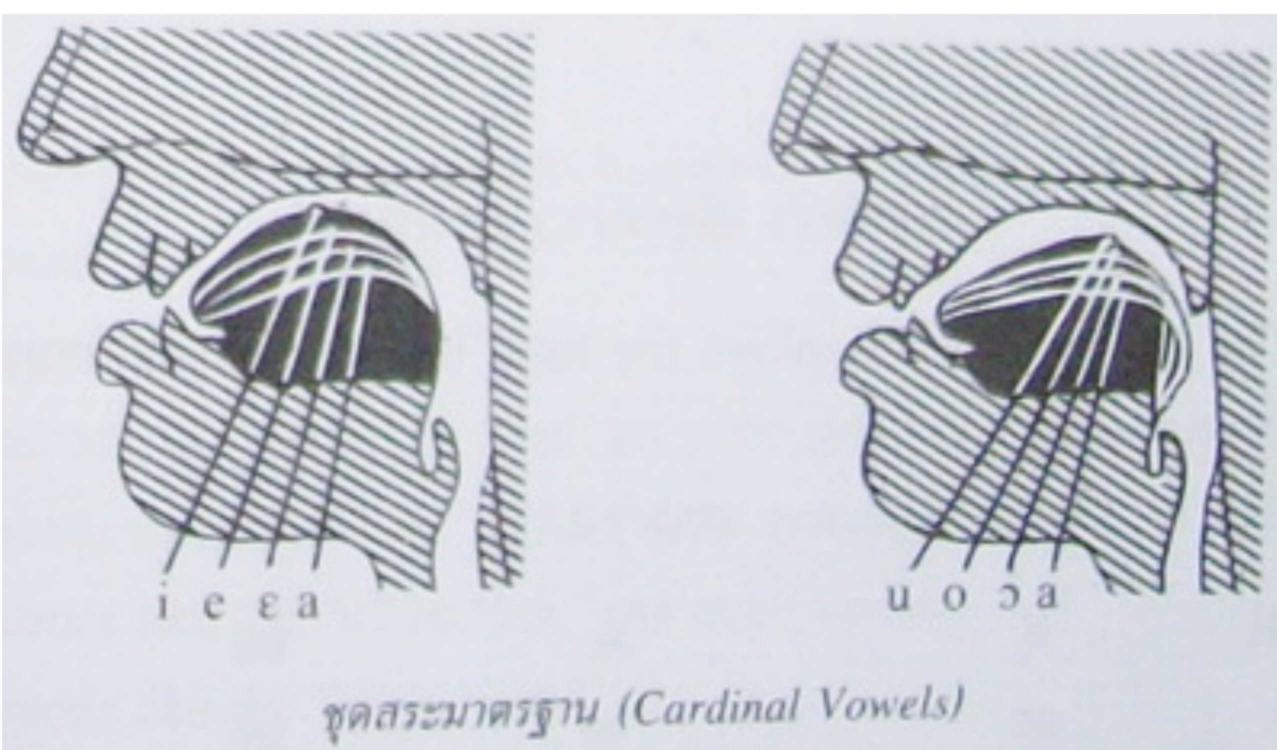
Linguistics Background: IPA

CONSONANTS (PULMONIC) © 2005 IPA

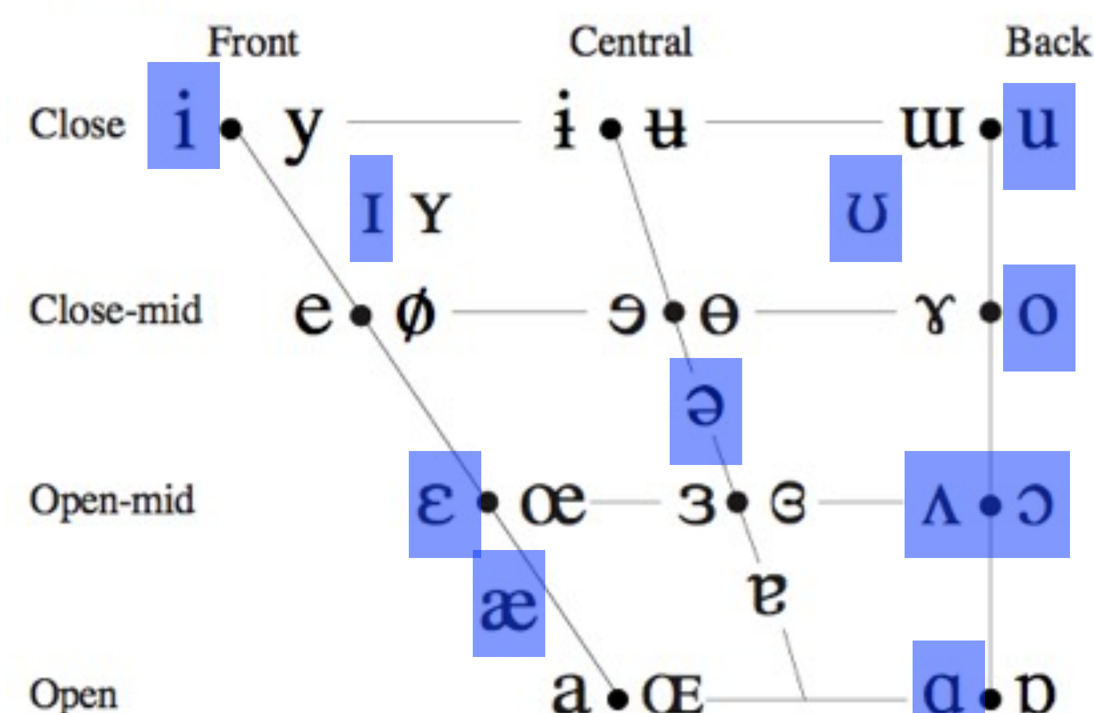
	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal	Glottal
Plosive	p b			t d		ʈ ɖ	c ɟ	k g	q ɢ		ʔ
Nasal	m	ɱ		n		ɳ	ɲ	ŋ	ɴ		
Trill	ʙ			r					ʀ		
Tap or Flap		ⱱ		ɾ		ɽ					
Fricative	ɸ β	f v	θ ð	s z	ʃ ʒ	ʂ ʐ	ç ʝ	x ɣ	χ ʁ	ħ ʕ	h ɦ
Lateral fricative				ɬ ɮ							
Approximant		ʋ		ɹ		ɻ	j	ɰ			
Lateral approximant				l		ɭ	ʎ	ʟ			

Where symbols appear in pairs, the one to the right represents a voiced consonant. Shaded areas denote articulations judged impossible.





VOWELS



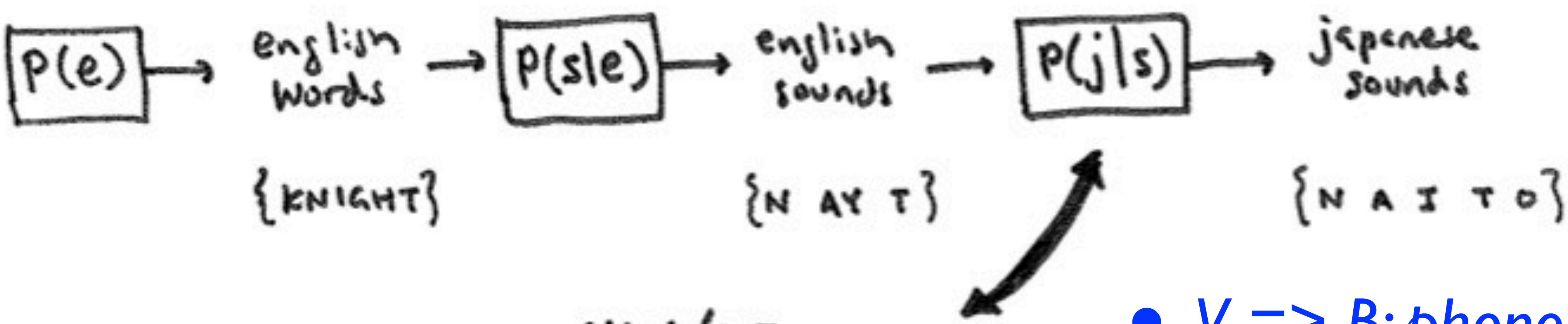
Where symbols appear in pairs, the one to the right represents a rounded vowel.

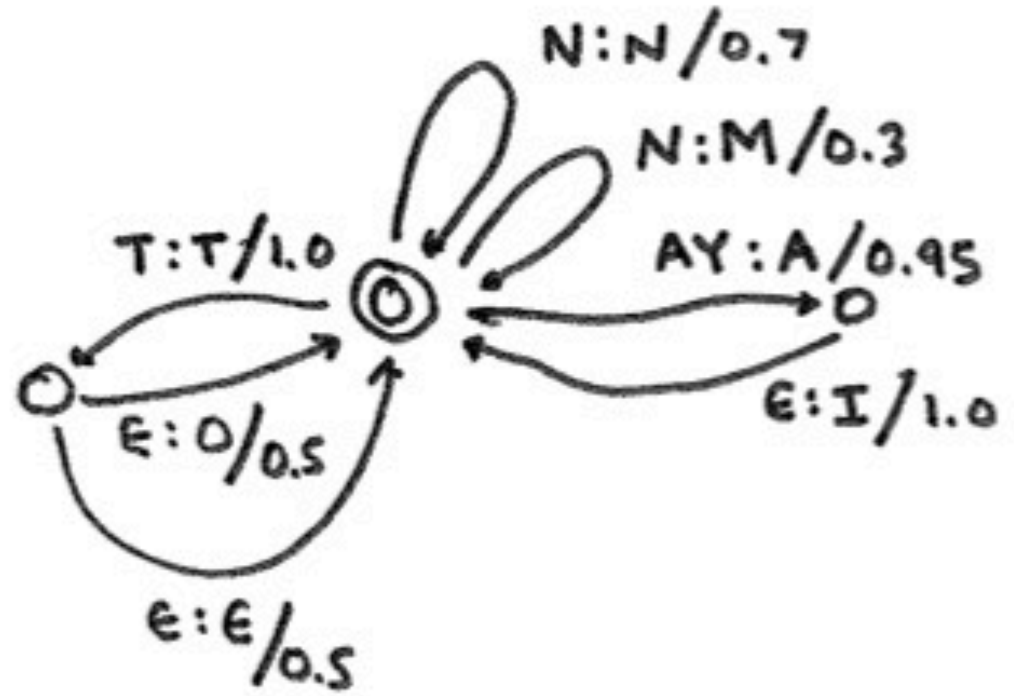
(hw2) From Sound *s* to Spelling *e*

- input: HH EH L OW B EH R
- output: H E L L O B E A R *or* H E L O B A R E ?
- $p(e) \Rightarrow e \Rightarrow p(s|e) \Rightarrow s$
- $p(w) \Rightarrow w \Rightarrow p(e|w) \Rightarrow e \Rightarrow p(s|e) \Rightarrow s$
- $p(w) \Rightarrow w \Rightarrow p(s|w) \Rightarrow s$
- $e \Leftarrow p(e|s) \Leftarrow s \Leftarrow p(s)$
- $w \Leftarrow p(w|e) \Leftarrow e \Leftarrow p(e|s) \Leftarrow s \Leftarrow p(s)$
- $w \Leftarrow p(w|s) \Leftarrow s \Leftarrow p(s)$
- what else?

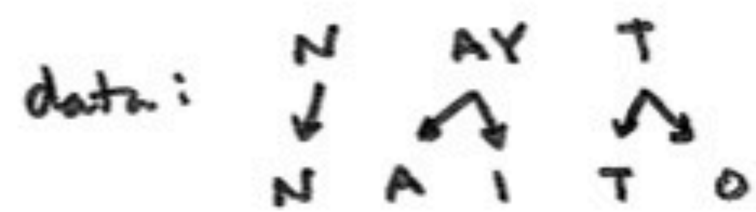
```
echo 'HH EH L OW' | carmel -sliOEQk 50 epron-espell.wfst
espell-eword.wfst eword.wfsa
```

Example: Transliteration





- *V => B: phoneme inventory mismatch*
- *T=>T O: phonotactic constraint*



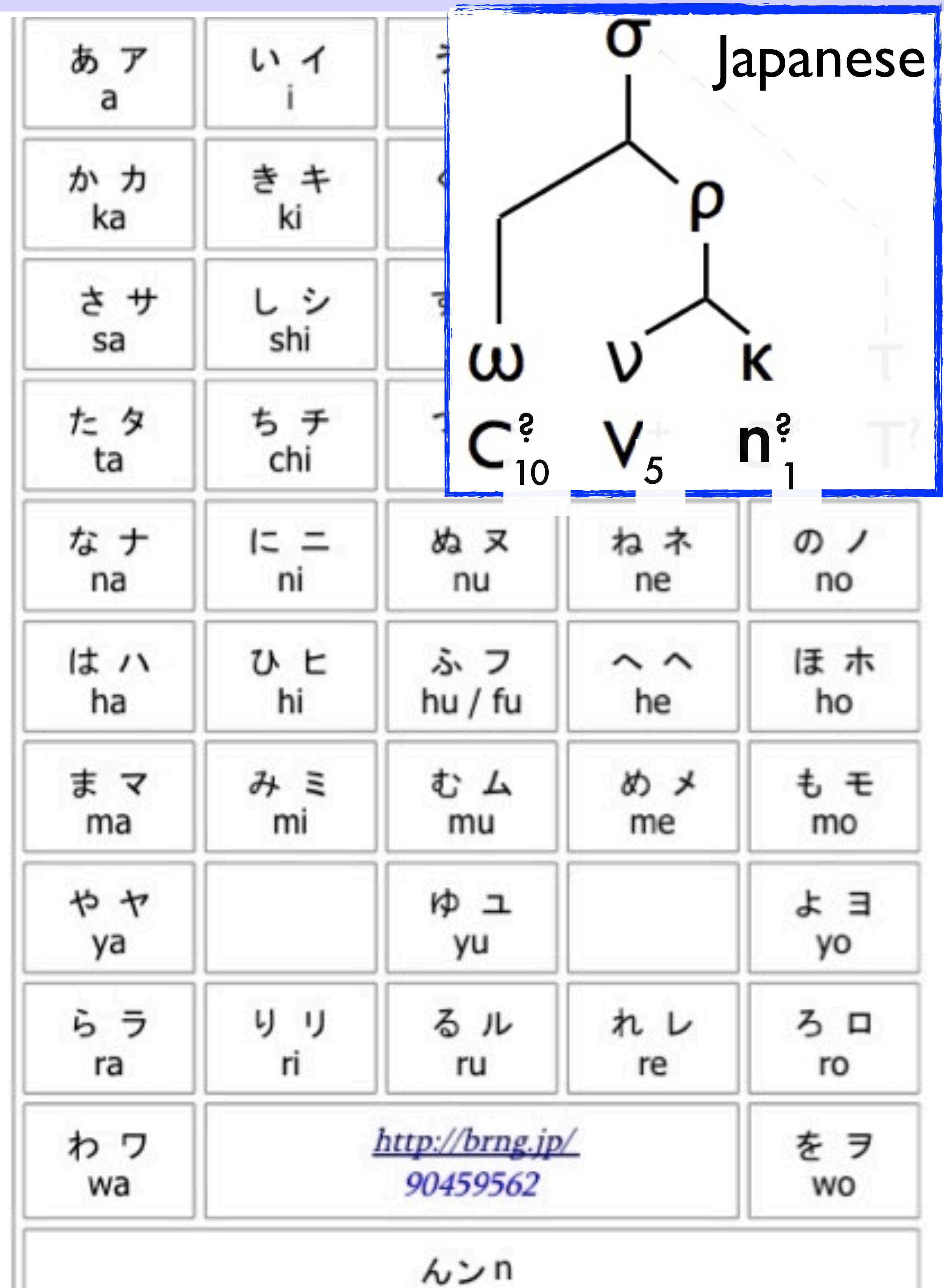
- KEVIN KNIGHT => KH EH VH IH N N AY T
  K E *B* I N N A I *T O*
  ケ ビ ン ナ イ ト

Japanese 101 (writing systems)

- Japanese writing system has four components
  - Kanji (Chinese chars): nouns, verb/adj stems, CJKV names
    - 日本 "Japan" 东京 "Tokyo" 电车 "train" 食べる "eat [inf.]"
  - Syllabaries
    - Hiragana: function words (e.g. particles), suffices
      - で de ("at") か ka (question) 食べました"ate"
    - Katakana: transliterated foreign words/names
      - コーヒー koohii ("coffee")
  - Romaji (Latin alphabet): auxiliary purposes

Why Japanese uses Syllabaries

- all syllables are:
  [consonant] + vowel + [nasal *n*]
- 10 C x 5 V = 50 syllables
  - plus some variations
- also possible for Mandarin
- other languages have many more syllables: use *alphabets*
  - alphabet = 10+5; syllabary = 10x5
- read the Writing Systems tutorial from course page!



Japanese Phonemes (too few sounds!)

Eng

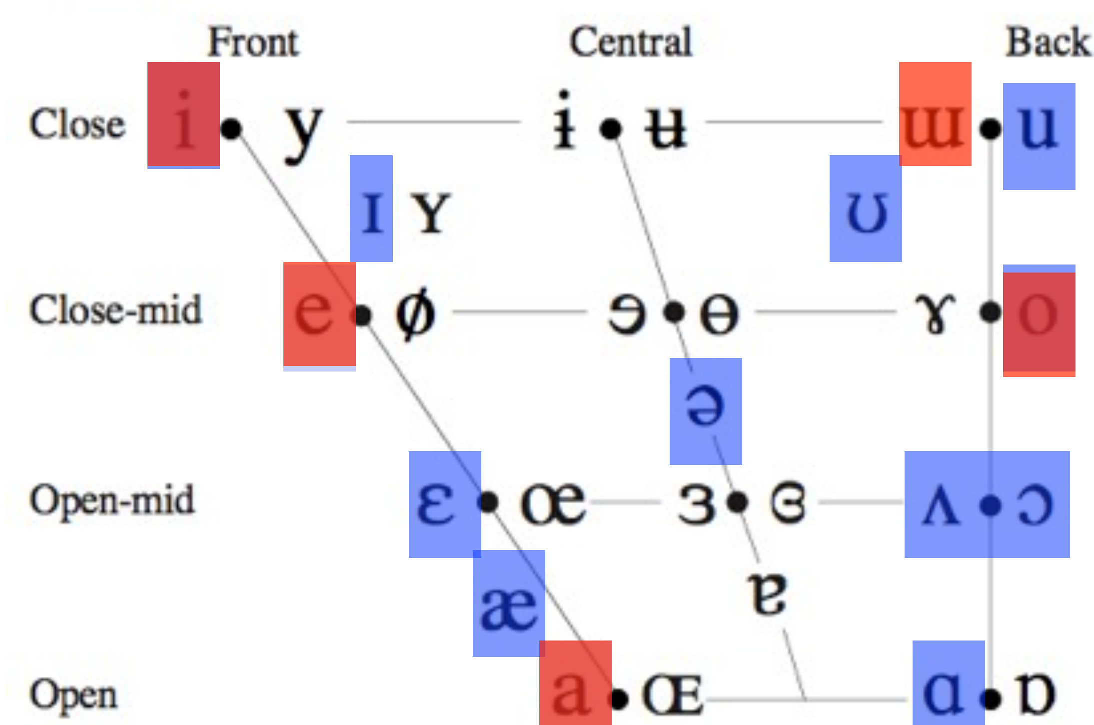
Jap

CONSONANTS (PULMONIC)

	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal	Glottal
Plosive	p b			t d		ʈ ɖ	c ɟ	k g	q ɢ		ʔ
Nasal	m	ɱ		n		ɳ	ɲ	ŋ	ɴ		
Trill	ʙ			r					ʀ		
Tap or Flap		ⱱ		ɾ		ɽ					
Fricative	ɸ β	f v	θ ð	s z	ʃ ʒ	ʂ ʐ	ç ʝ	x ɣ	χ ʁ	ħ ʕ	h ɦ
Lateral fricative				ɬ ɮ							
Approximant		ʋ		ɹ		ɻ	j	ɰ			
Lateral approximant				l		ɭ	ʎ	ʟ			

Where symbols appear in pairs, the one to the right represents a voiced consonant. Shaded areas denote articulations judged impossible.

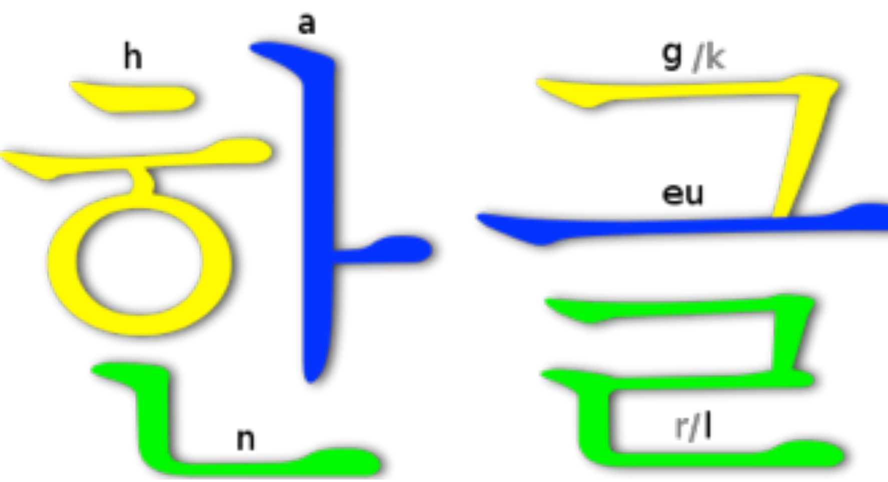
VOWELS

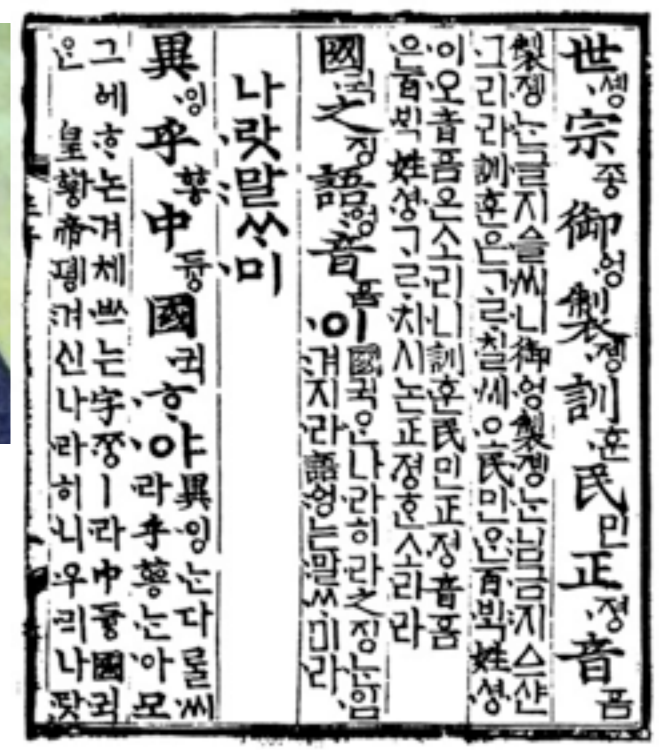


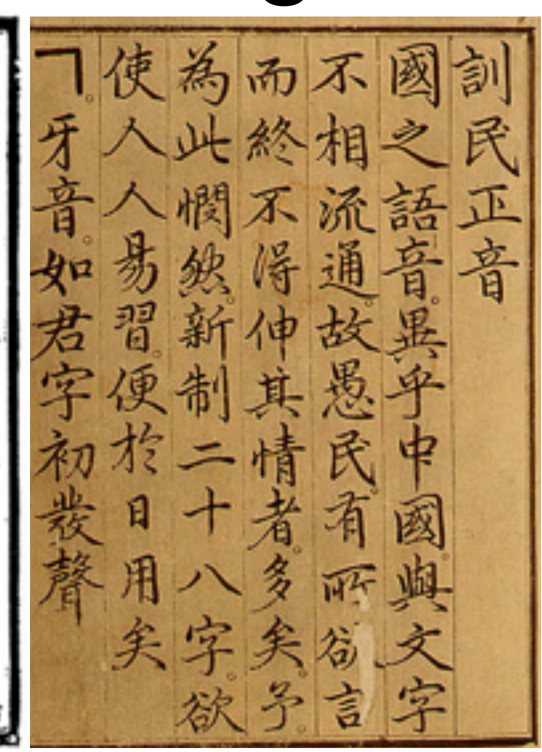
Where symbols appear in pairs, the one to the right represents a rounded vowel.

Aside: Is Korean a Syllabary?

- A: Hangul is not a syllabary, but a "featural alphabet"
  - a special alphabet where shapes encode phonological features
- the inventor of Hangul (c. 1440s) was the first real linguist







- 14 consonants: ㄱ *g*, ㄴ *n*, ㄷ *d*, ㄹ *l/r*, ㅁ *m*, ㅂ *b*, ㅅ *s*, ㅇ *null/ng*, ㅈ *j*, ㅊ *ch*, ㅋ *k*, ㅌ *t*, ㅍ *p*, ㅎ *h*
- 5 double consonants: ㄲ *kk*, ㄸ *tt*, ㅃ *pp*, ㅆ *ss*, ㅉ *jj*
- 11 consonant clusters: ㄳ *gs*, ㄵ *nj*, ㄶ *nh*, ㄺ *lg*, ㄻ *lm*, ㄼ *lb*, ㄽ *ls*, ㄾ *lt*, ㄿ *lp*, ㅀ *lh*, ㅄ *bs*
- 6 vowel letters: 아 *a*, 어 *eo*, ㅗ *o*, ㅜ *u*, ㅡ *eu*, 이 *i*
- 4 iotized vowels (with a *y*): 야 *ya*, 여 *yeo*, ㅛ *yo*, ㅠ *yu*
- 5 (iotized) diphthongs: ㅐ *ae*, ㅒ *yae*, ㅔ *e*, ㅖ *ye*, ㅢ *ui*
- 6 vowels and diphthongs with a *w*: ㅘ *wa*, ㅙ *wae*, ㅚ *oe*, ㅝ *wo*, ㅞ *we*, ㅟ *wi*

Q: 강남 스타일 = ?

Katakana Transliteration Examples

- コンピューター
- ko n py u - ta -
- kompyuutaa (uu=û)
- computer

- アイスクリーム
- a i su ku ri - mu
- aisukuriimu
- ice cream

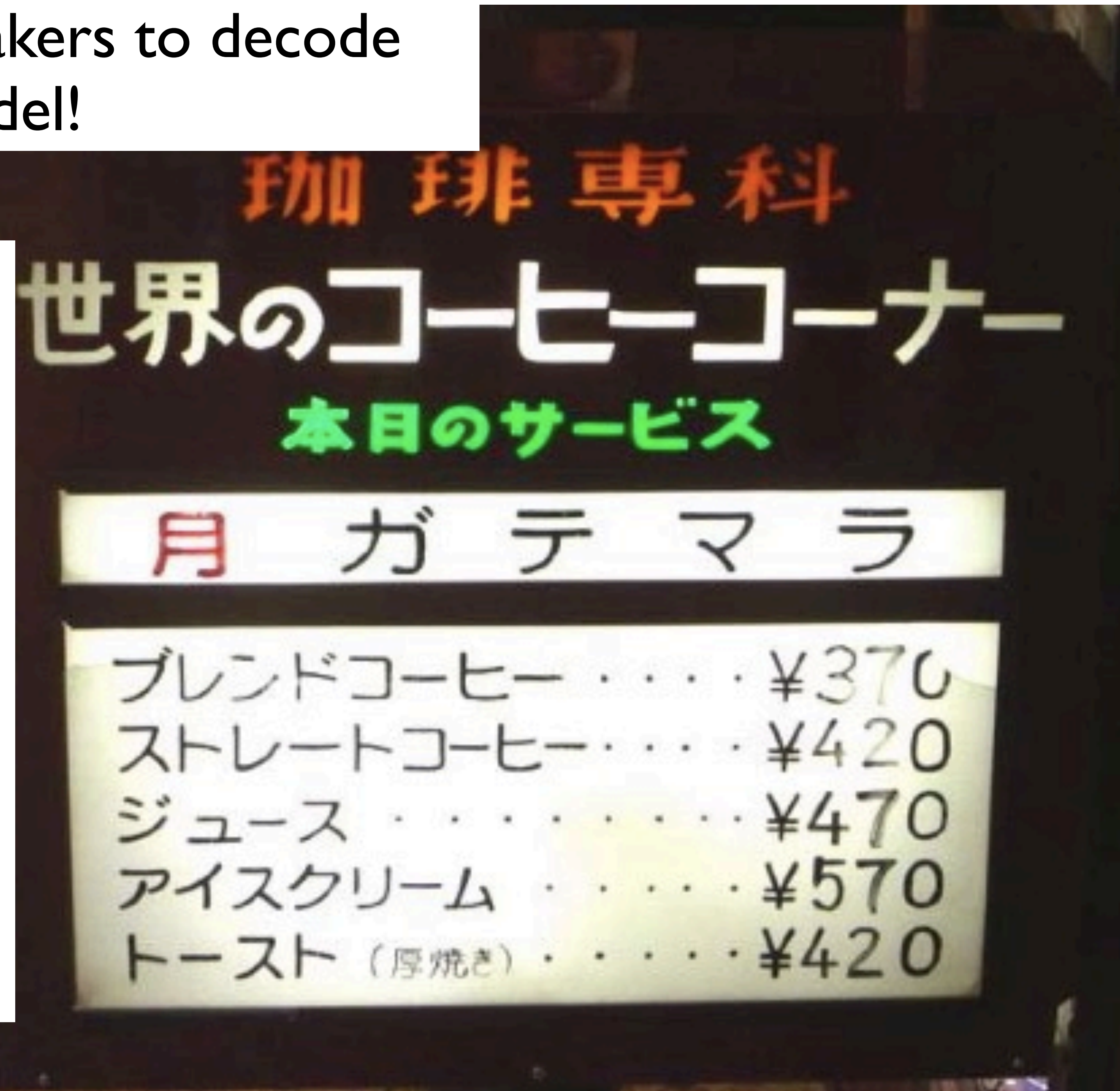
- アンドリュー・ビタビ
- andoryuubitabi
- Andrew Viterbi

- ヨーグルト
- yo - gu ru to
- yogurt

Katakana on Streets of Tokyo

Japanese just transliterates almost everything
(even though its syllable inventory is really small...)
but... it is quite easy for English speakers to decode
.... if you have a good language model!

from Knight & Sproat 09



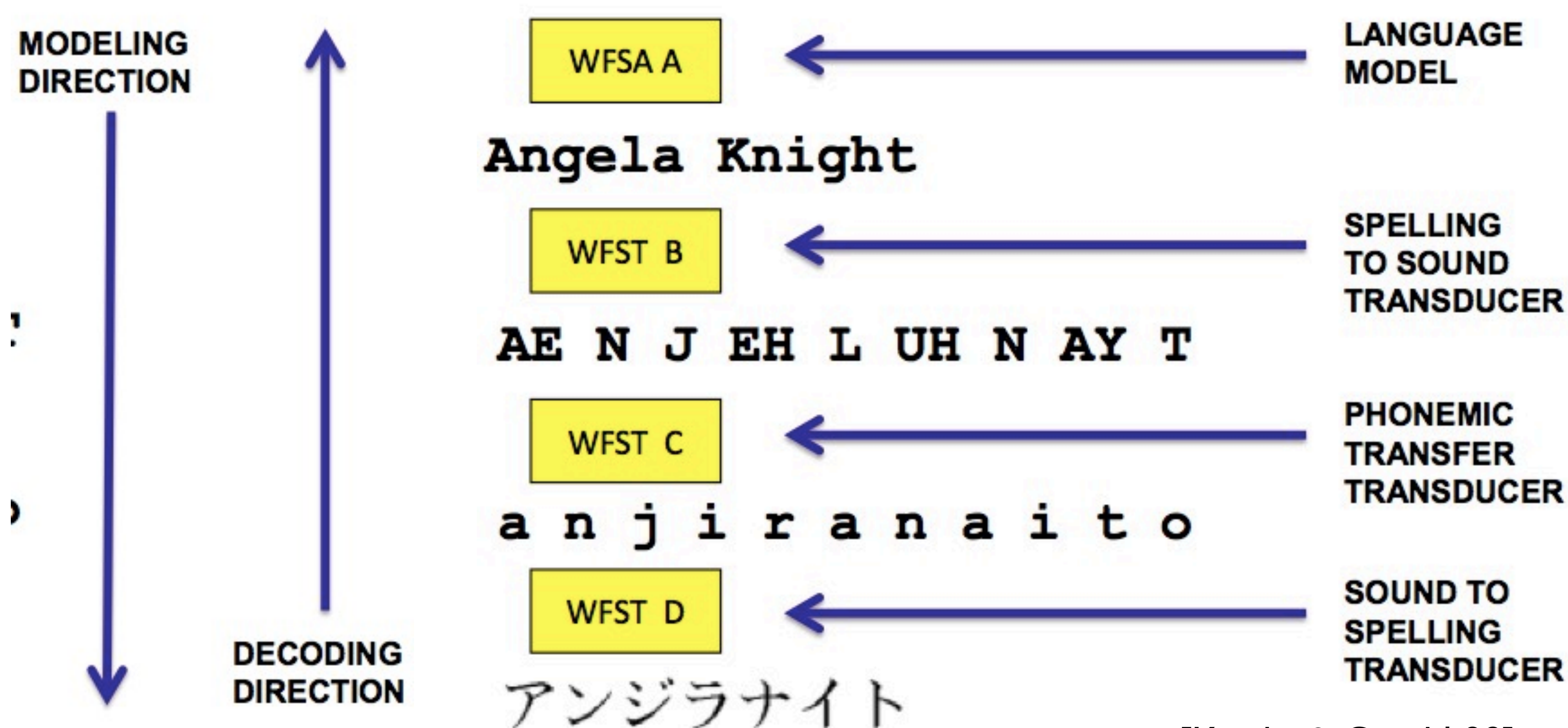
koohiikoonaa	coffee corner
saabisu	service
bulendokoohii	blend coffee
sutoreetokoohii	straight coffee
juusu	juice
aisukuriimu	ice cream
toosuto	toast

More Japanese Transliterations

rapputoppu	ラプトプ	laptop
bideoteepu	ビデオテープ	video tape
shoppingusentaa	ショッピングセンター	shopping center
shiitoberuto	シートベルト	seat belt
chairudoshiito	チャイルトシート	child seat
andoryuubitabi	アンドリュー・ビタビ	Andrew Viterbi
bitabiarugorizumu	ビタビアルゴリズム	Viterbi Algorithm

(hw2) Katakana => English

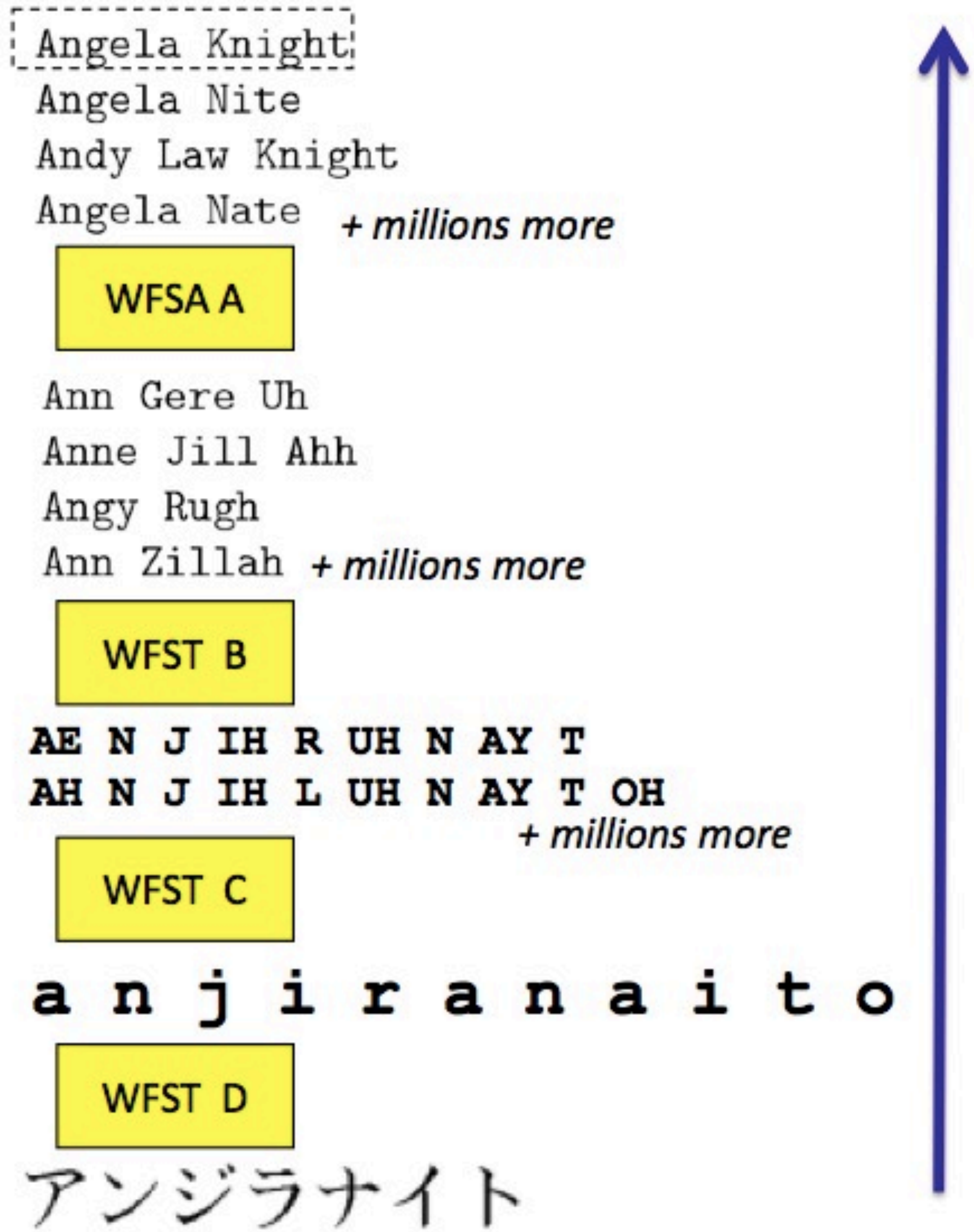
- your job in HW2: decode Japanese Katakana words (transcribed in Romaji) back to English words
  - koohiikoonaa => coffee corner



[Knight & Graehl 98]

(hw2) Katakana => English

- Decoding (HW3)
  - really decipherment!
- what about duplicate strings?
  - from different paths in WFST!
  - n-best cruching, or...
  - weighted determinisation
    - see extra reading on course website for Mohri+Riley paper

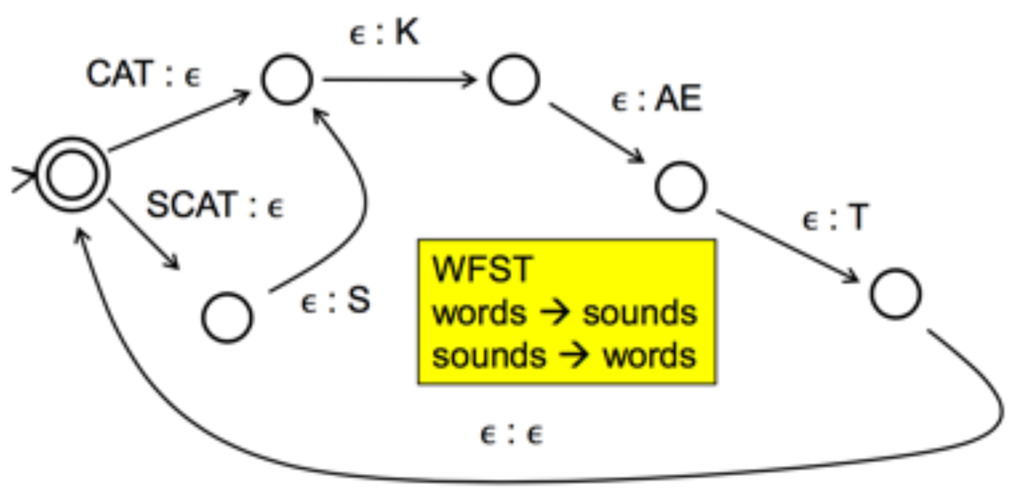


[Knight & Graehl 98]

How to Learn p(e|w) and p(j|e)?

- Such a system captures an infinite relation of <sound-sequence, writing-sequence> pairs.

HW2
eword-epron.data



Learning Sequence Transformation Probabilities

HW2
epron-jpron.data
(MLE)

HW3
Viterbi decoding

Ideal training data:

etc

P(n | M) = 0.5
P(m u | M) = 0.5

need much more data, of course

HW4
epron-jpron.data
(EM)

Actual training data:

L AE M P
r a n p u

S T IY M
s u t i i m u

etc

Automatically align string pairs using the unsupervised Expectation-Maximization (EM) algorithm.

String Transformations

- General Framework for many NLP problems
- Examples

  - Part-of-Speech Tagging
  - Spelling Correction (Edit Distance)
  - Word Segmentation
  - Transliteration, Sound/Spelling Conversion, Morphology
  - Chunking (Shallow Parsing)
  - Beyond Finite-State Models (i.e., tree transformations)
    - Summarization, Translation, Parsing, Information Retrieval, ...
- Algorithms: Viterbi (both max and sum)

Example 2: Part-of-Speech Tagging

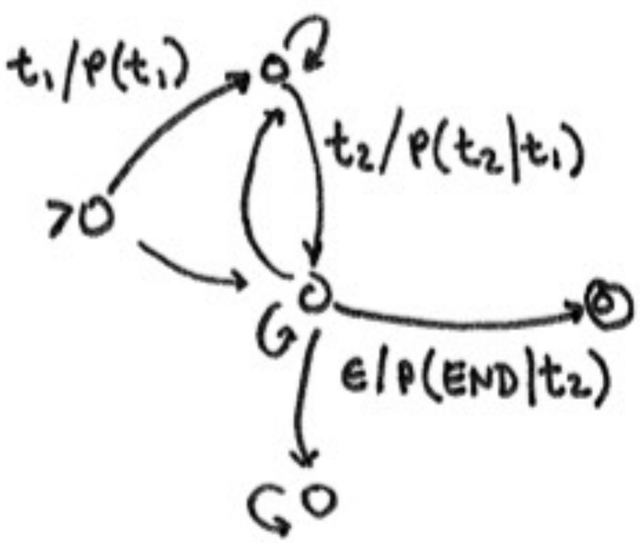
$$P(t \cdots t \mid w \cdots w)$$

$$\sim P(t \cdots t) \cdot P(w \cdots w \mid t \cdots t)$$

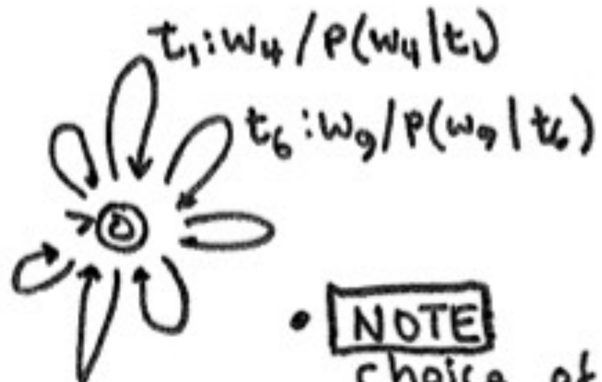
$$\sim \underbrace{P(t_1) \cdot P(t_2 \mid t_1) \cdots P(t_n \mid t_{n-1})}_{\text{local grammar preference}} \cdot \underbrace{P(w_1 \mid t_1) \cdots P(w_n \mid t_n)}_{\text{lexical preference}}$$

- use tag bigram as a language model
- channel model is context-indep.

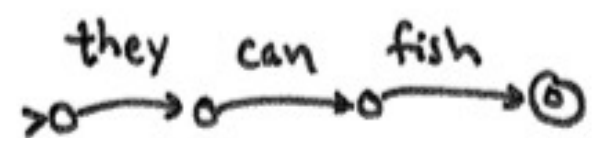
source



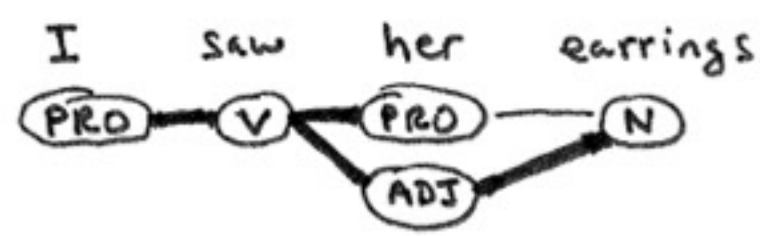
channel



new string



- NOTE
  choice of tag is influenced by both left & right context

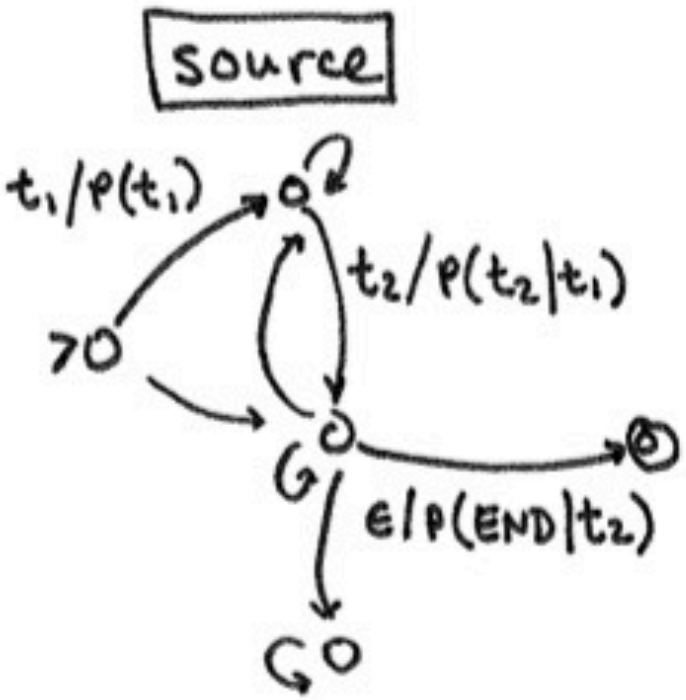


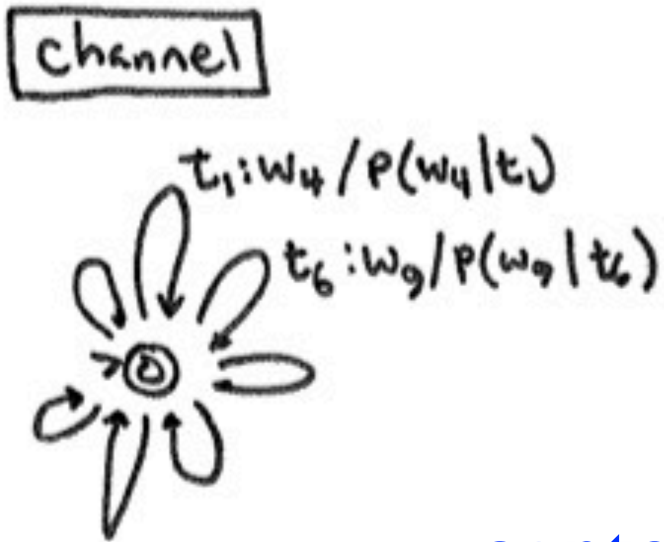
- NOTE
  influences can theoretically be long ranging

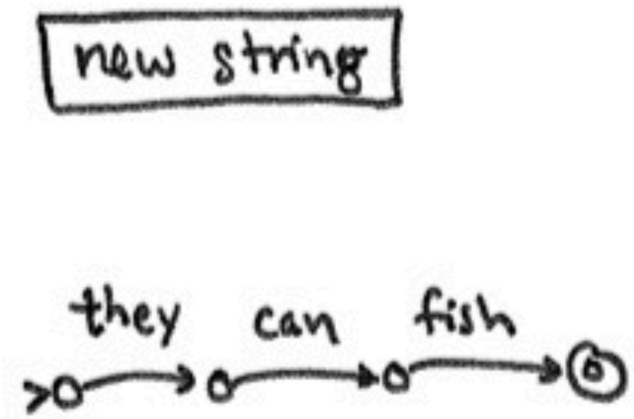
Work out the compositions

- if you want to implement Viterbi...
- case 1: language model is a tag unigram model
  - $p(t...t) = p(t_1)p(t_2) ... p(t_n)$
  - how many states do you get?
- case 1: language model is a tag bigram model
  - $p(t...t) = p(t_1)p(t_2 | t_1) ... p(t_n | t_{n-1})$
  - how many states do you get?
- case 3: language model is a tag trigram model...

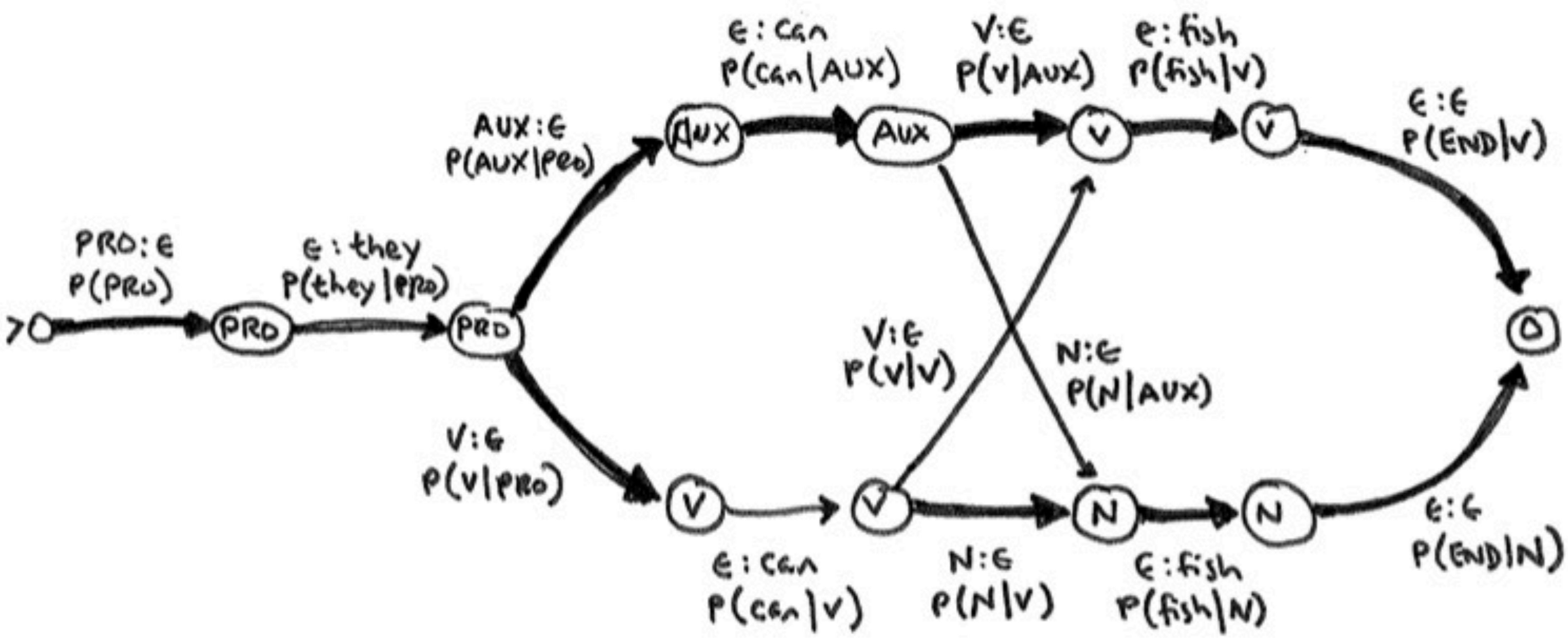
The case of bigram model





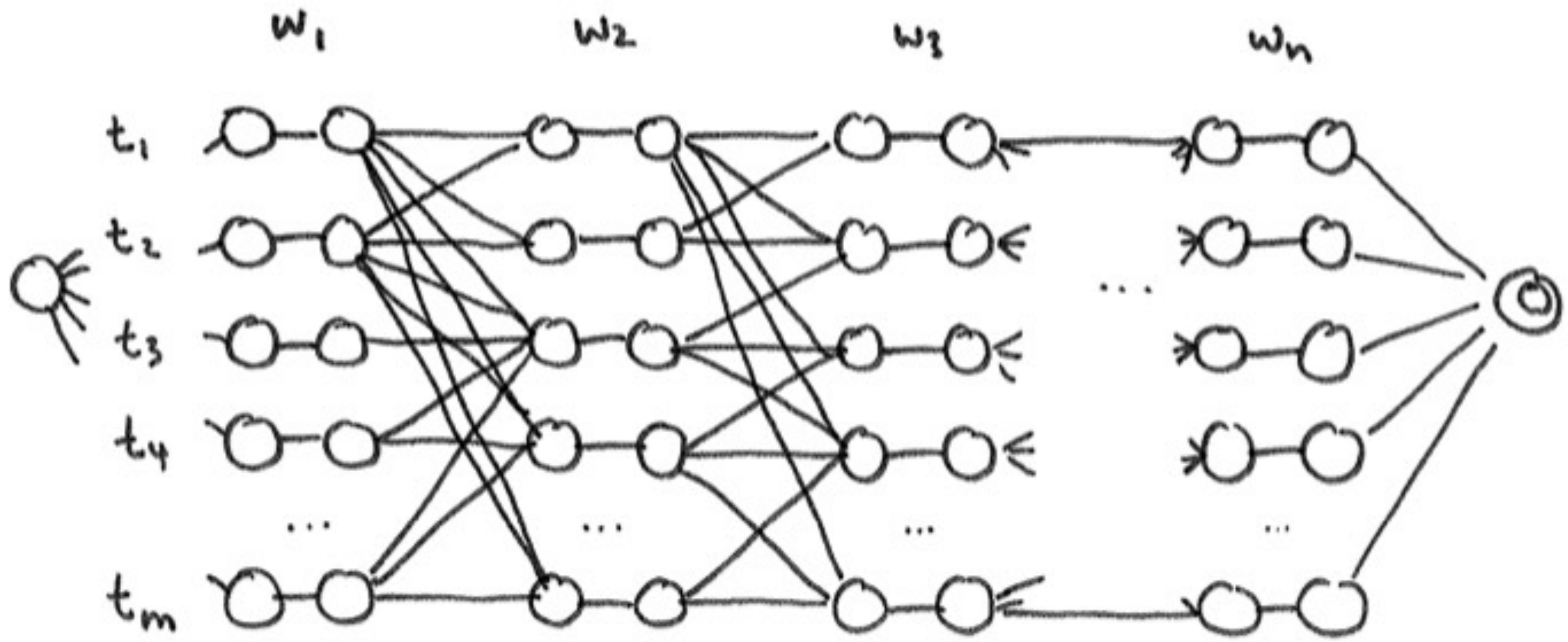


context-dependence (from LM)
propagates left and right!

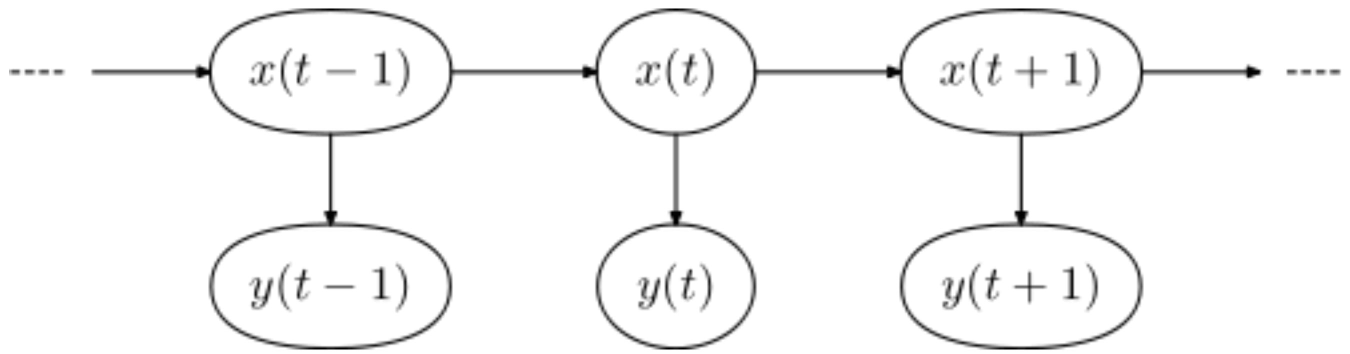


In general...

- bigram LM with context-independent CM
  - $O(n\ m)$ states after composition
- g-gram LM with context-independent CM
  - $O(n\ m^{g-1})$ states after composition
  - the g-gram LM itself has $O(m^{g-1})$ states



HMM Representation



- HMM representation is not explicit about the search
  - "hidden states" have choices over "variables"
  - in FST composition, paths/states are explicitly drawn

Viterbi for argmax

Viterbi search for $\underset{t \cdots t}{\operatorname{argmax}}\ P(t \cdots t) \cdot P(w \cdots w \mid t \cdots t)$:

```
for j = 1 to m
    Q[1,j] = P(t_j) · P(w_1 | t_j)

for i = 2 to n
    for j = 1 to m
        Q[i,j] = 0
        best-pred[i,j] = 0
        best-score = -∞
        for k = 1 to m
            r = P(t_j | t_k) · P(w_i | t_j) · Q[i-1,k]
            if r > best-score
                best-score = r
                best-pred[i,j] = k
                Q[i,j] = r

final-best = 0
final-score = -∞
for j = 1 to m
    if Q[n,j] > final-score
        final-score = Q[n,j]
        final-best = j

print t_final-best
current = final-best
for i = n-1 down to 1
    current = best-prev[i+1, current]
    print t_current
```

Q[i,j] = cost of shortest path ending with word i getting assigned tag j.

sets back pointers

prints best tags in reverse order

how about unigram?

Python implementation

Complete this Python code implementing the Viterbi algorithm for part-of-speech tagging. It should print a list of word/tag pairs, e.g. `[('a', 'D'), ('can', 'N'), ('can', 'A'), ('can', 'V'), ('a', 'D'), ('can', 'N')]`.

Q: what about top-down recursive + memoization?

```
from collections import defaultdict

best = defaultdict(lambda : defaultdict(float))
best[0]["<s>"] = 1
back = defaultdict(dict)

words = "<s> a can can can a can </s>".split()

tags = {"a": ["D"], "can": ["N", "A", "V"], "</s>": ["</s>"]}  # possible tags for each word
ptag = {"D": {"N": 1}, "V": {"</s>": 0.5, "D":0.5}, ... }      # ptag[x][y] = p(y | x)
pword = {"D": {"a": 0.5}, "N": {"can": 0.1}, ...       }      # pword[x][w] = p(w | x)

for i, word in enumerate(words[1:], 1):                       # i=1,2...; word=a,can,...
    for tag in tags[word]:
        for prev in best[i-1] :
            if tag in ptag[prev] :
                score = best[i-1][prev] * ptag[prev][tag] * pword[tag][word]
                if score > best[i][tag]:
                    best[i][tag] = score
                    back[i][tag] = prev

def backtrack(i, tag):
    if i == 0:
        return []
    return backtrack(i-1, back[i][tag]) + [(words[i], tag)]

print backtrack(len(words)-1, "</s>")[:-1]
```

Viterbi Tagging Example

prob of \ given	START	PRO	V	N	AUX
END		.1	.1	.1	.1
PRO	.6				
V	.05	.6		.2	.9
N	.3		.9	.7	
AUX	.05				

prob of \ given	PRO	V	N	AUX
they	.07			
can		10^{-5}	10^{-4}	.21
fish		10^{-4}	10^{-4}	

Q1. why is this table *not* normalized?

Q2. is "fish" equally likely to be a V or N?

Q3: how to train p(w|t)?

	they	can	fish
PRO	Q = P(PRO|START)·P(they|PRO) = .6 · .07 = .042	Q = 0 P(PRO|PRO) = 0 P(can|PRO) = 0 !	Q = 0
V	Q = 0 P(they|V) = 0	Q = max⟨.042 · .6 · 10^{-5}⟩ = .000000252 bp = PRO	Q: max⟨.000000252 · 0 · 10^{-4}, .002646 · .9 · 10^{-4}⟩ = .00000023814 bp = AUX
N	Q = 0	Q = 0 P(N|PRO) = 0	Q = max⟨.000000252 · .9 · 10^{-4}, .002646 · 0 · 10^{-4}⟩ = .0000000000002268 bp = V
AUX	Q = 0	Q = max⟨.042 · .3 · .21⟩ = .002646 bp = PRO	Q = 0

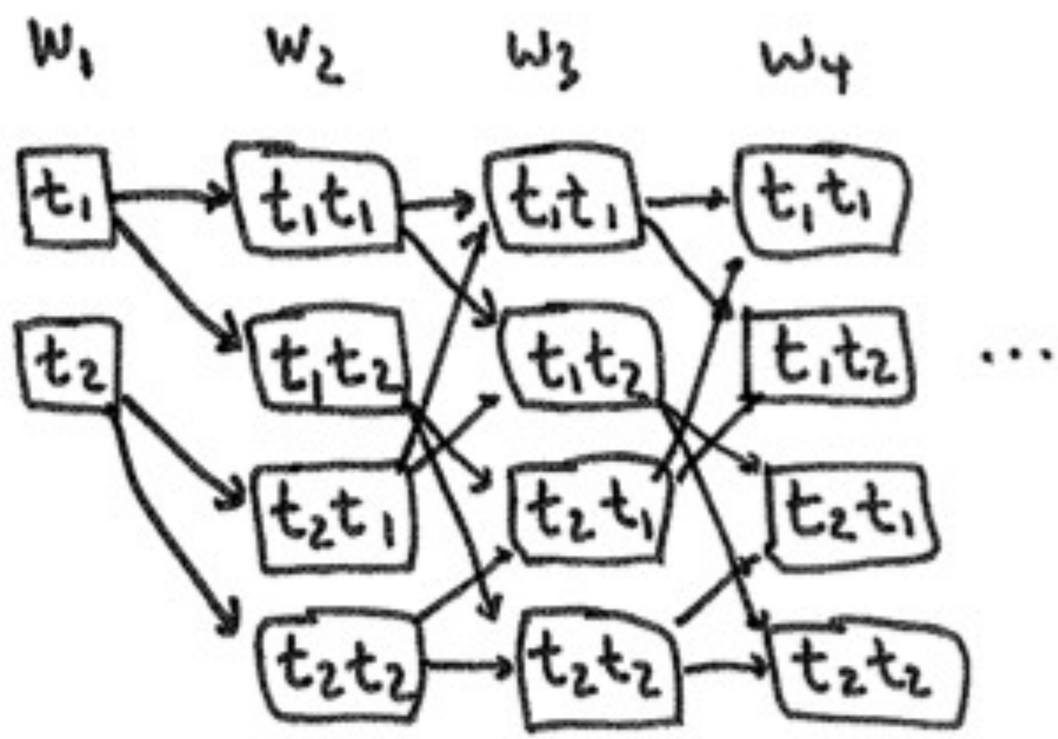
bp = V

$$Q[1,j] = P(t_j \mid \text{START}) \cdot P(w_1 \mid t_j)$$

$$Q[i,j] = \max_k Q[i-1,k] \cdot P(t_j \mid t_k) \cdot P(w_i \mid t_j)$$

Trigram HMM

```
for j = 1 to m
    Q_1[1,j] = ...

for j = 1 to m
    for j2 = 1 to m
        Q[2,j,j2] = ...

for i = 3 to n
    for j = 1 to m
        for j2 = 1 to m
            Q[i,j,j2] = 0
            best-pred[i,j,j2] = 0
            best-score = -∞
            for k = 1 to m
                r = P(t_j2 | t_k t_j) · P(w_i | t_j2) · Q[i-1, k, j]
                if r > best-score ...
```



time complexity: $O(nT^3)$

in general: $O(nT^g)$ for g-gram

A Side Note on Normalization

$\boxed{\text{NOTE}}$

final-best gives $P(t \cdots t) \cdot P(w \cdots w \mid t \cdots t)$

but this is not the same as $P(t \cdots t \mid w \cdots w)$

e.g. suppose there is only one $t \cdots t$ (all words unambiguous)

then $P(t \cdots t \mid w \cdots w) = 1$

need to divide

$$P(t \cdots t \mid w \cdots w) = \frac{P(t \cdots t) \cdot P(w \cdots w \mid t \cdots t)}{P(w \cdots w)} = \frac{P(t \cdots t) \cdot P(w \cdots w \mid t \cdots t)}{\sum_{t \cdots t} P(t \cdots t) \cdot P(w \cdots w \mid t \cdots t)}$$

how to compute the normalization factor?

Forward (sum instead of max)

Forward search: $\sum_t P(t) \cdot P(w|t) = P(w)$

$\alpha[1,j] = P(t_j|\text{START}) \cdot P(w_1|t_j)$

$\alpha[i,j] = \text{sum}_k \; \alpha[i-1,k] \cdot P(t_j|t_k) \cdot P(w_i|t_j)$

no back pointer

$P(w) = \text{sum}_k \; \alpha[n,k]$

"Forward" procedure for $P(w \cdots w)$

$\alpha[i,j]$ = costs of all paths ending w/ word w_i getting tag t_j (costs summed)

```
for j = 1 to m
  α[1,j] = P(t_j) · P(w_1|t_j)

for i = 2 to n
  for j = 1 to m
    α[i,j] = 0
    for k = 1 to m
      α[i,j] += P(t_j|t_k) · P(w_i|t_j) · α[i-1,k]

P(w···w) = 0
for j = 1 to m
  P(w···w) += α[n,j]
```

Forward vs. Argmax

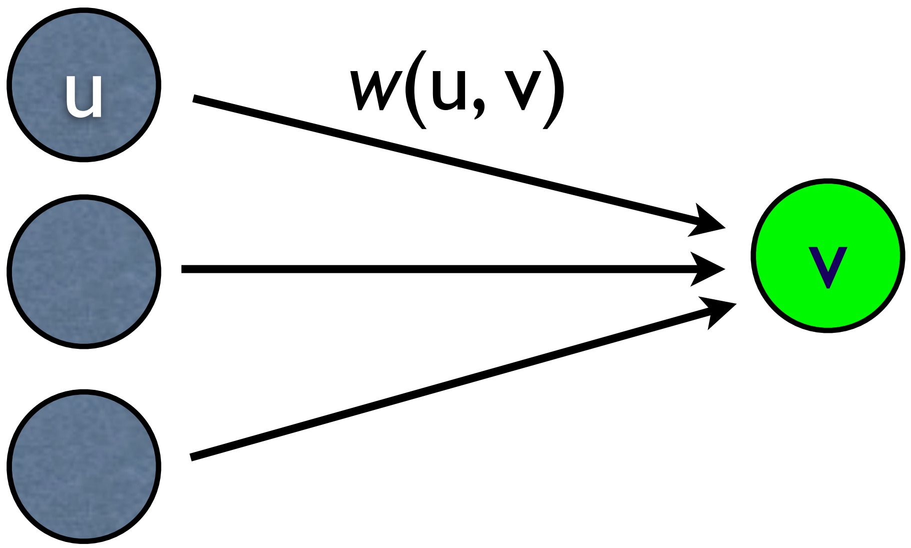
- same complexity, different semirings (+, x) vs (max, x)
- for g-gram LM with context-indep. CM
  - time complexity O(n m^{g})   space complexity O(n m^{g-1})

```
for j = 1 to m
    Q1[1,j] = ...

for j = 1 to m
    for j2 = 1 to m
        Q[2,j,j2] = ...

for i = 3 to n
    for j = 1 to m
        for j2 = 1 to m
            Q[i,j,j2] = 0
            best-pred[i,j,j2] = 0
            best-score = -∞
            for k = 1 to m
                r = P(t_j2 | t_k t_j) · P(w_i | t_j2) · Q[i-1, k, j]
                if r > best-score ...
```

$\boxed{O(nm^3) \text{ complexity}}$

Viterbi for DAGs with Semiring

1. topological sort $(A, \oplus, \otimes, \overline{0}, \overline{1})$
2. visit each vertex v in sorted order and do updates
   - for each incoming edge (u, v) in E
   - use d(u) to update d(v): $d(v) \oplus= d(u) \otimes w(u, v)$
   - key observation: d(u) is fixed to optimal at this time



see tutorial on DP from course page

- time complexity: O(V + E)

Example: Word Segmentation

- you noticed that Japanese (e.g., Katakana) is written *without* spaces between words
  - in order to guess the English you also do segmentation
  - e.g. アイスクリーム => アイス クリーム => ice cream
  - how about "gaaruhurendo" and "shingururuumu" ?
- this is an even more important issue in Chinese
  - 南京市长江大桥
- also in other East Asian Languages
- also in English: sounds => words (speech recognition)

What if English were written as Chinese...

- thisisacoursetaughtinthefallsemesterofthisyearatusc
- actually, Latin used to be written exactly like this!
  - “scripta continua” => “interpuncts” (center dots) =>
- this might be a final project topic (on the easier side)

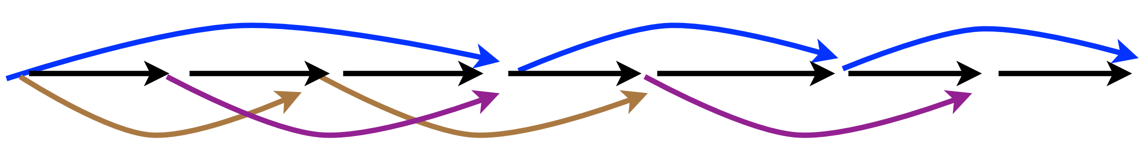
Chinese Word Segmentation

民主
min-zhu
people-dominate
"democracy"

Google
this was 5 years ago.
now Google is
good at segmentation!

江泽民 主席
jiang-ze-min zhu-xi
... - ... - *people* *dominate-podium*
"President Jiang Zemin"

下 雨 天 地 面 积 水
xia yu tian di mian ji shui



graph search

下 雨 天 地 面 积 水

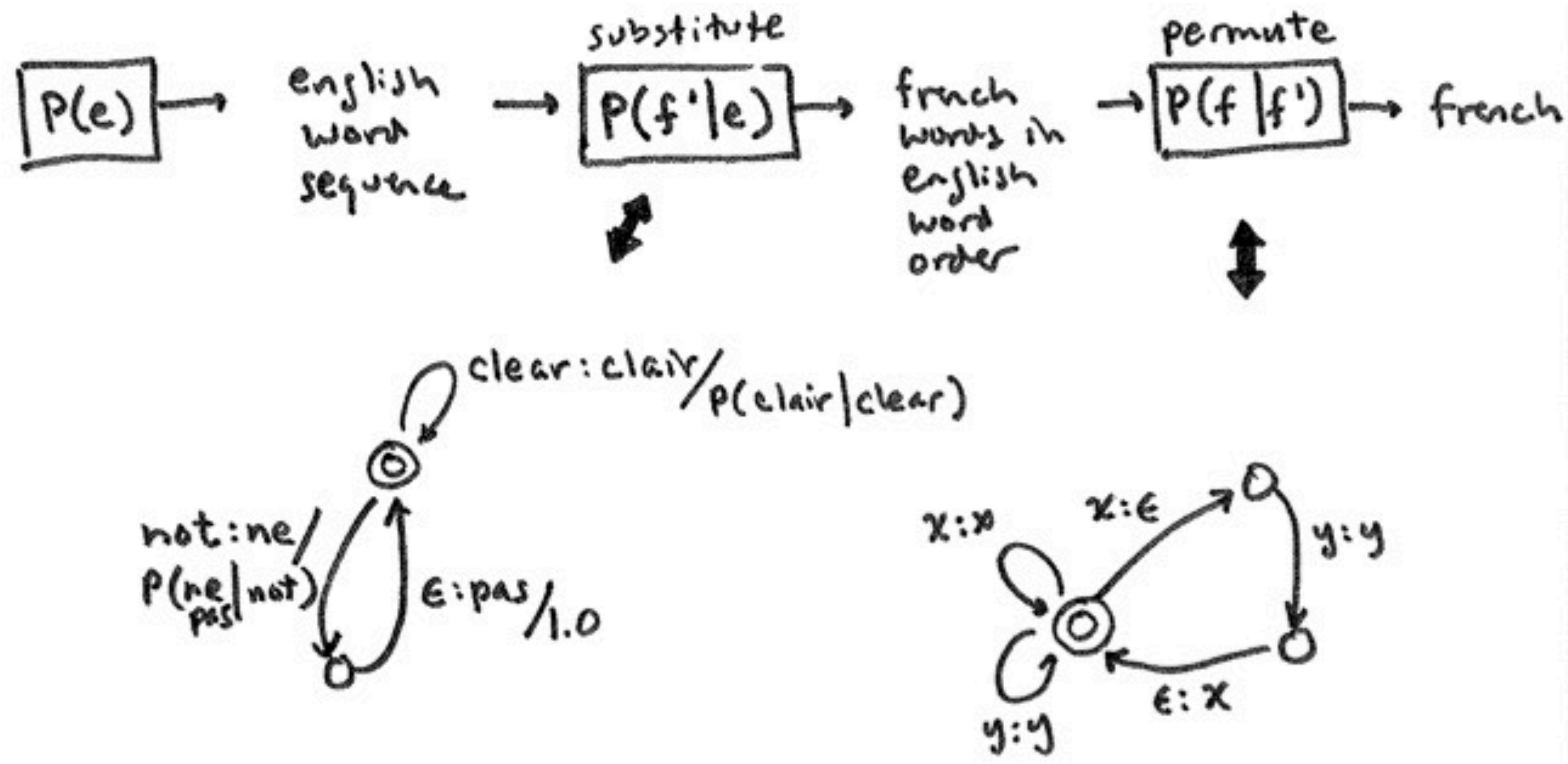
tagging problem

Word Segmentation Cascades

- a good idea for final project (Chinese/Japanese)

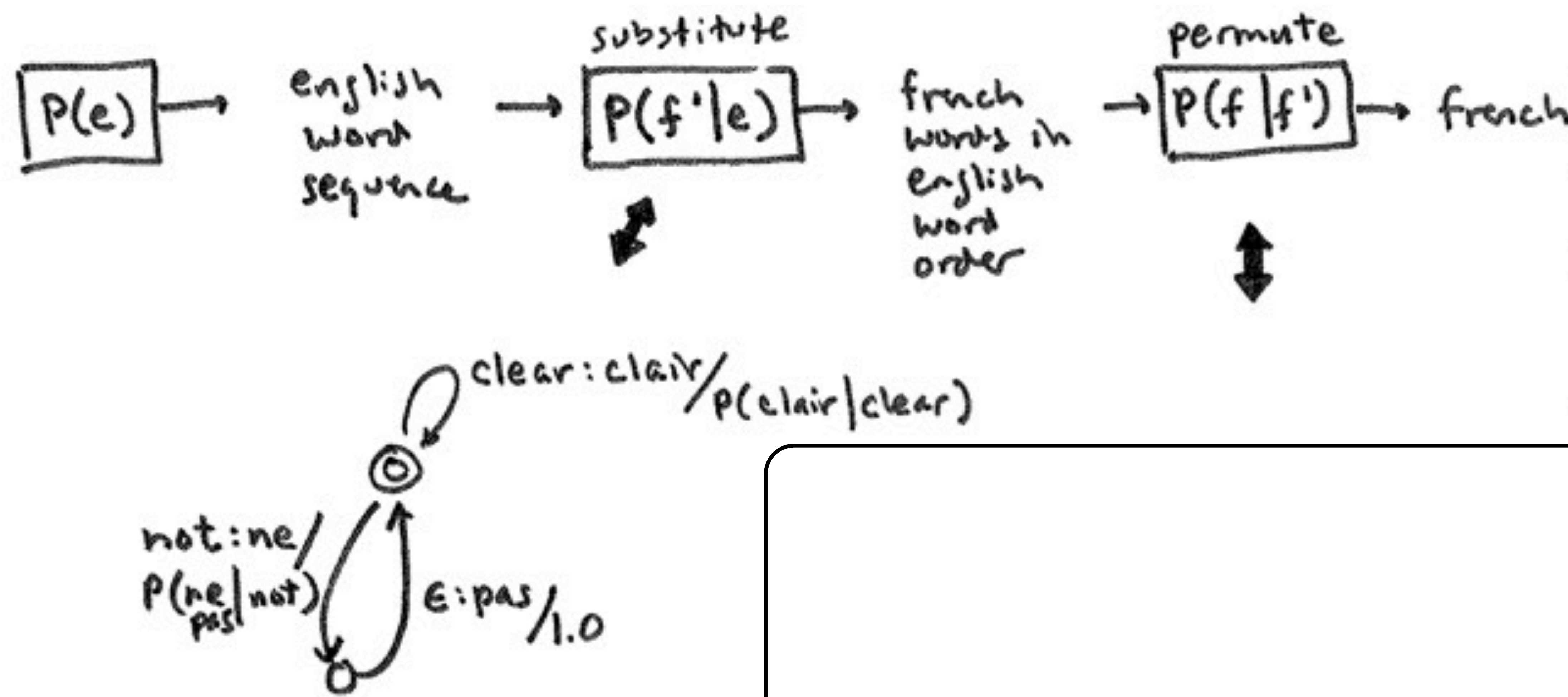
Machine Translation

- simplest model: word-substitution and permutation
- does it really work??

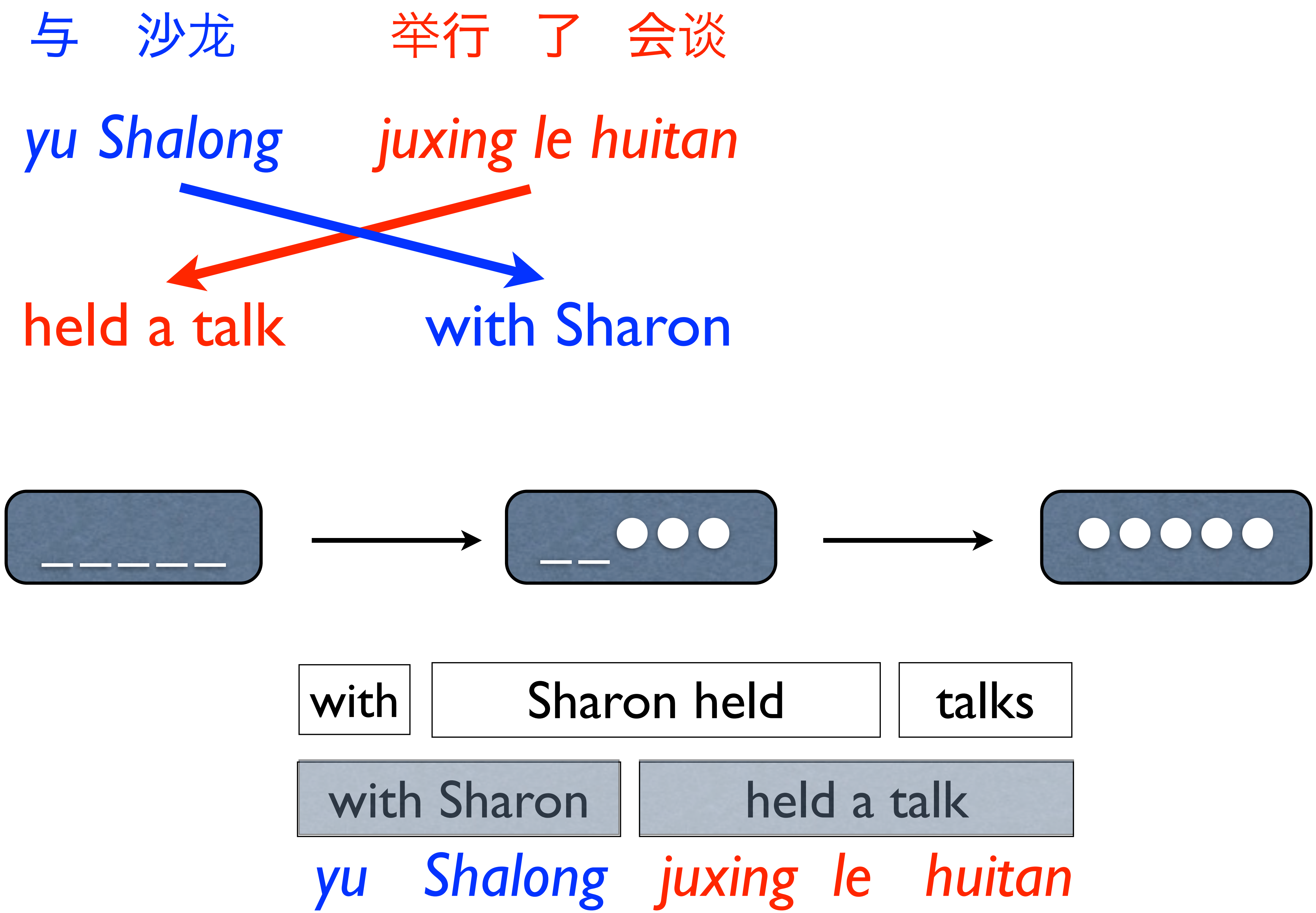


Machine Translation Permutation

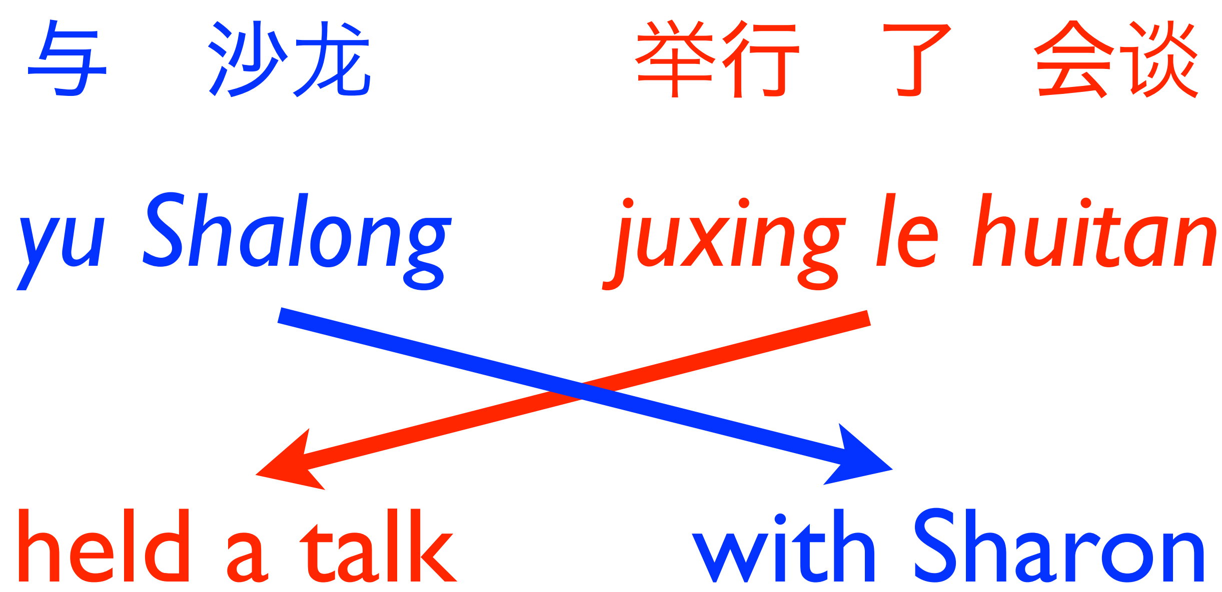
- how would you model permutation in FSTs?



Phrase-based Decoding



Phrase-based Decoding

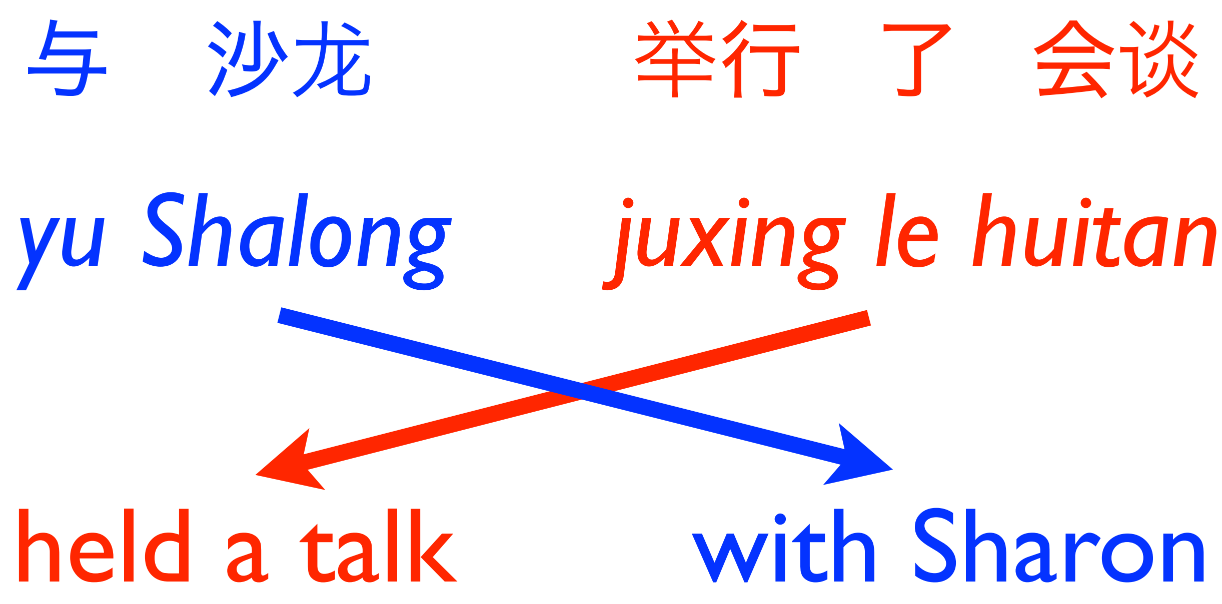


with | Sharon held | talks

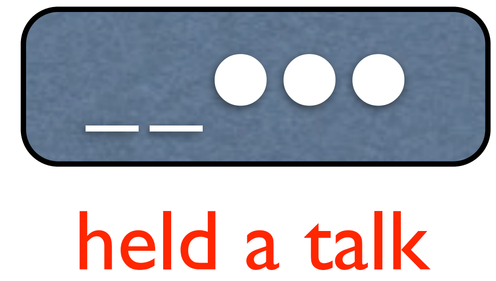
with Sharon | held a talk

yu Shalong juxing le huitan

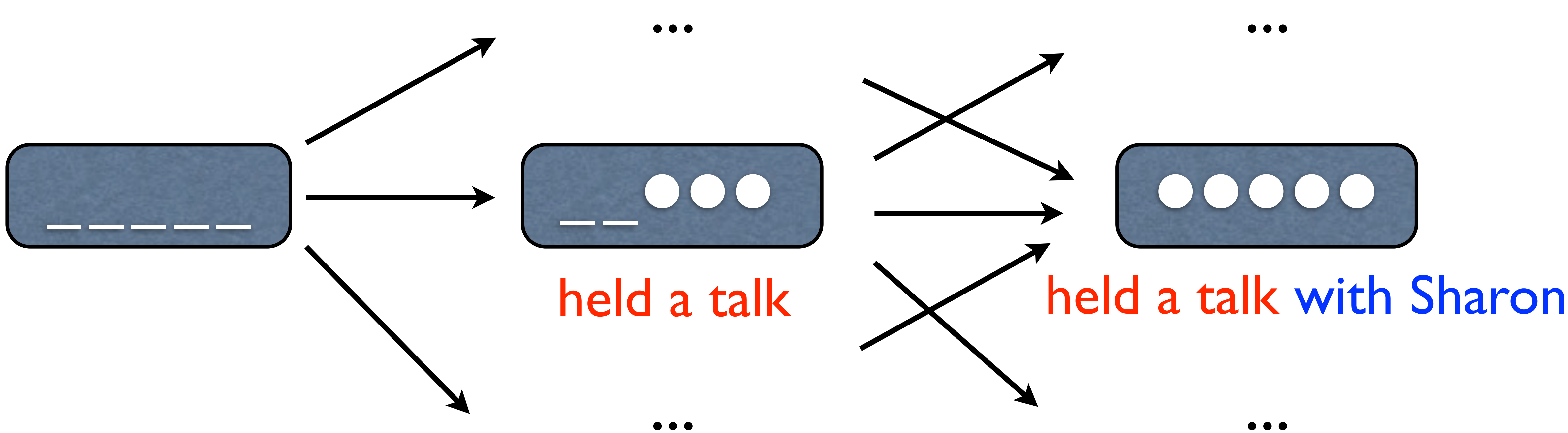
Phrase-based Decoding



source-side: coverage vector



target-side: grow hypotheses strictly left-to-right



space: $O(2^n)$, time: $O(2^n n^2)$ -- cf. traveling salesman problem

Phrase-based Cascades

- english LM => (english) => phrase substitutions (n^2) => (foreign phrases in english word order) => permutations (2^n)=> (foreign)
- a good idea for final project (on the harder end)
- wait, where does the phrase table come from?
  - => word-aligned english-foreign sentence pairs

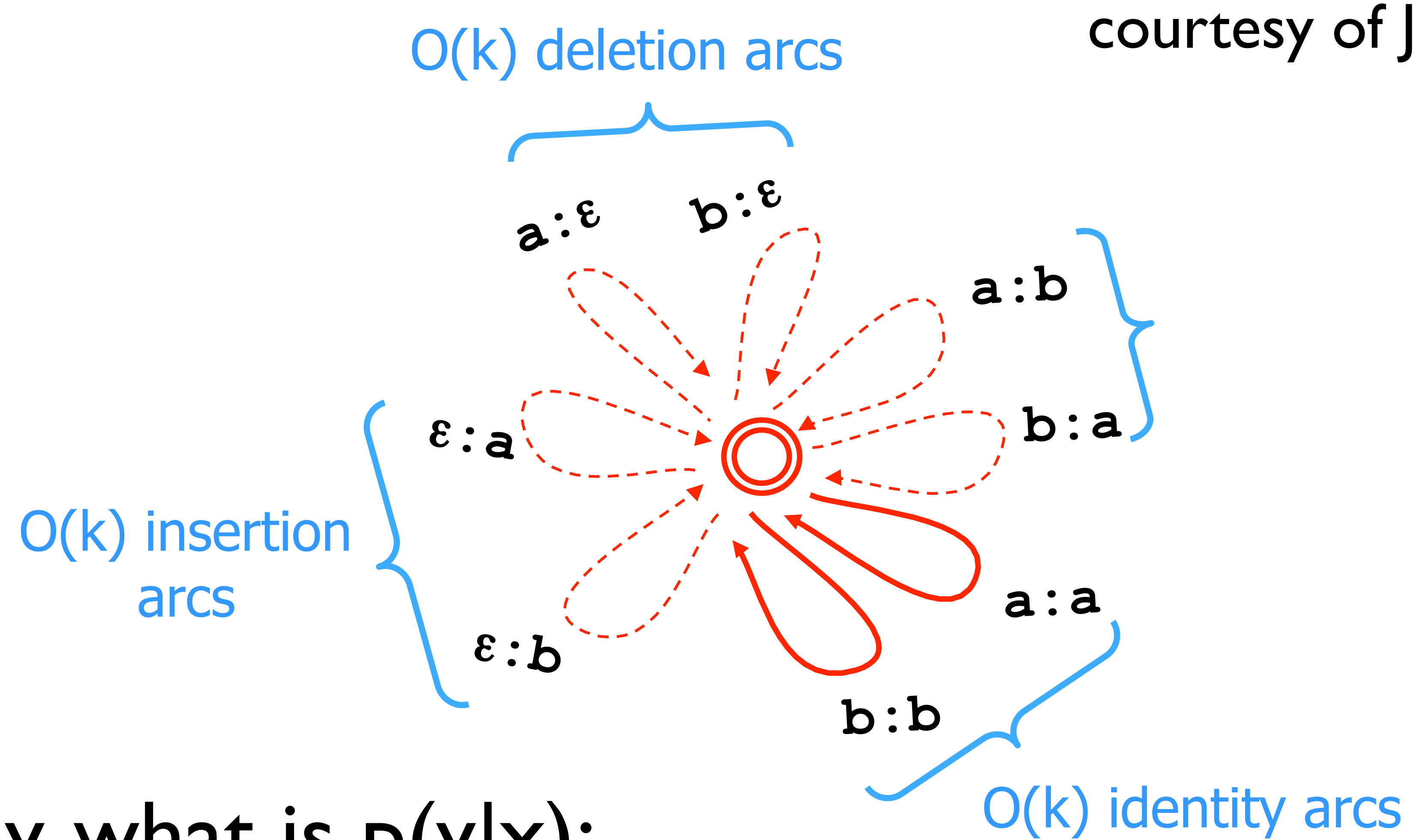
Traveling Salesman Problem & MT

- a classical NP-hard problem
  - goal: visit each city once and only once
- exponential-time dynamic programming
  - state: cities visited so far (bit-vector)
  - search in this $O(2^n)$ transformed graph
- MT: each city is a source-language word
  - restrictions in reordering can reduce complexity => distortion limit
  - => syntax-based MT

(Held and Karp, 1962; Knight, 1999)

Example: Edit Distance

courtesy of Jason Eisner



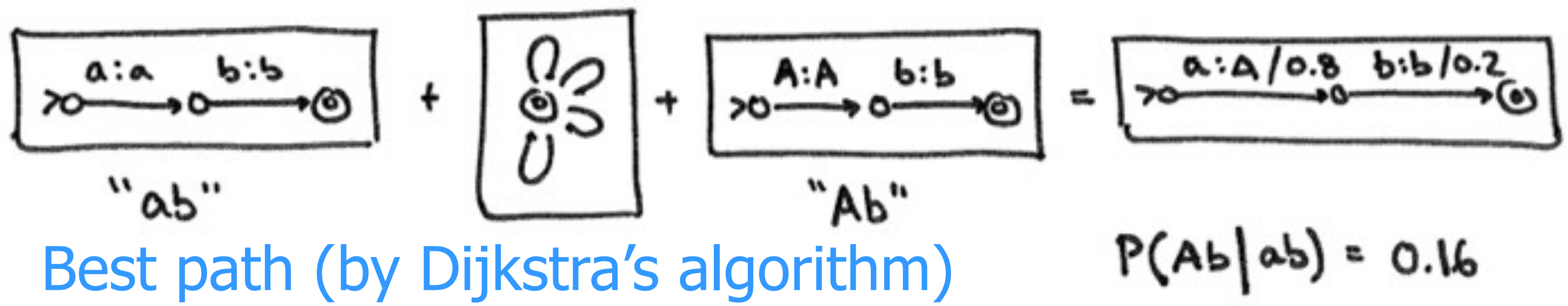
- a) given x, y, what is p(y|x);
- b) what is the most likely seq. of operations?
- c) given x, what is the most likely output y?
- d) given y, what is the most likely input x (with LM) ?

Edit Distance can model...

- part-of-speech tagging
- transliteration
- sound-spelling conversion
- word-segmentation

Given x and y...

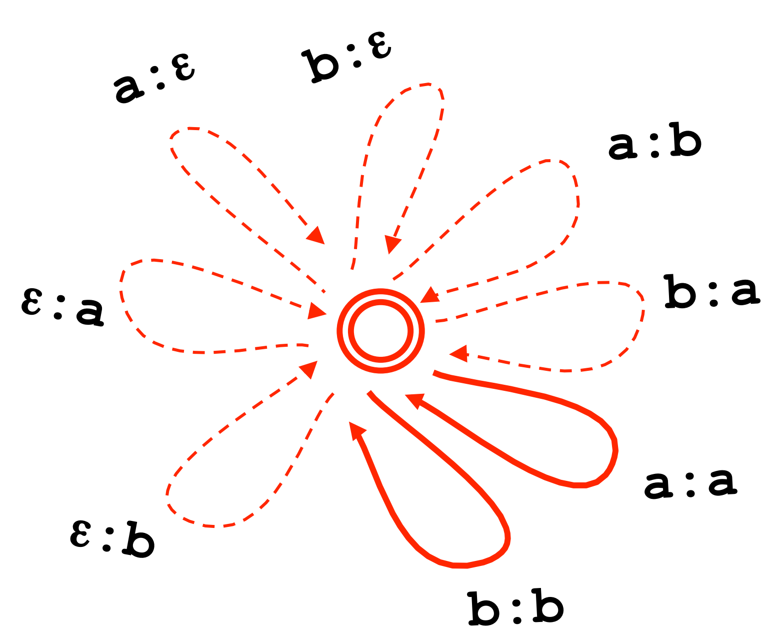
- given x, y: a) what is p(y | x)? (sum of all paths)
  b) what is the most likely conversion path?



Best path (by Dijkstra's algorithm)

clara

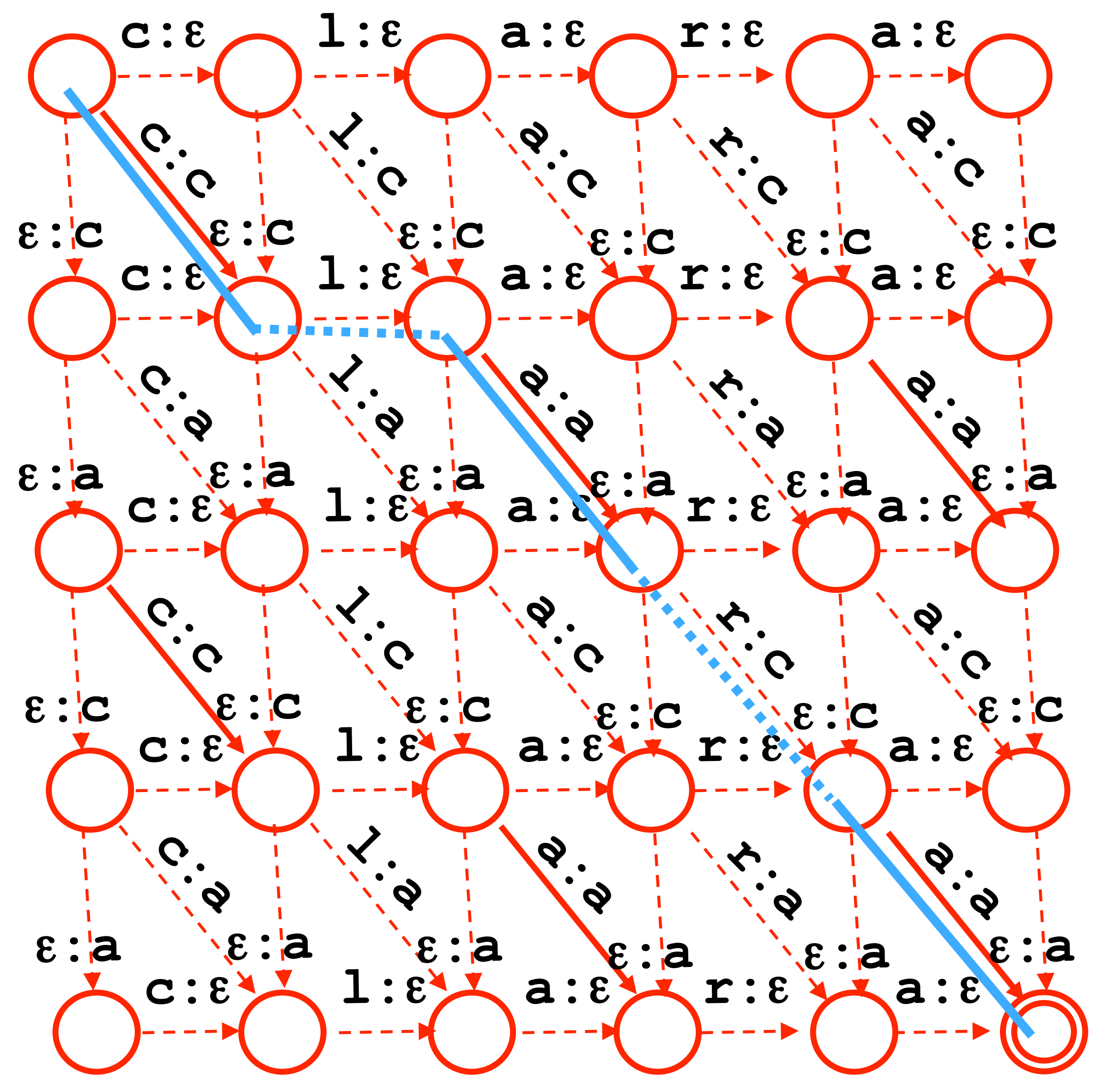
.o.



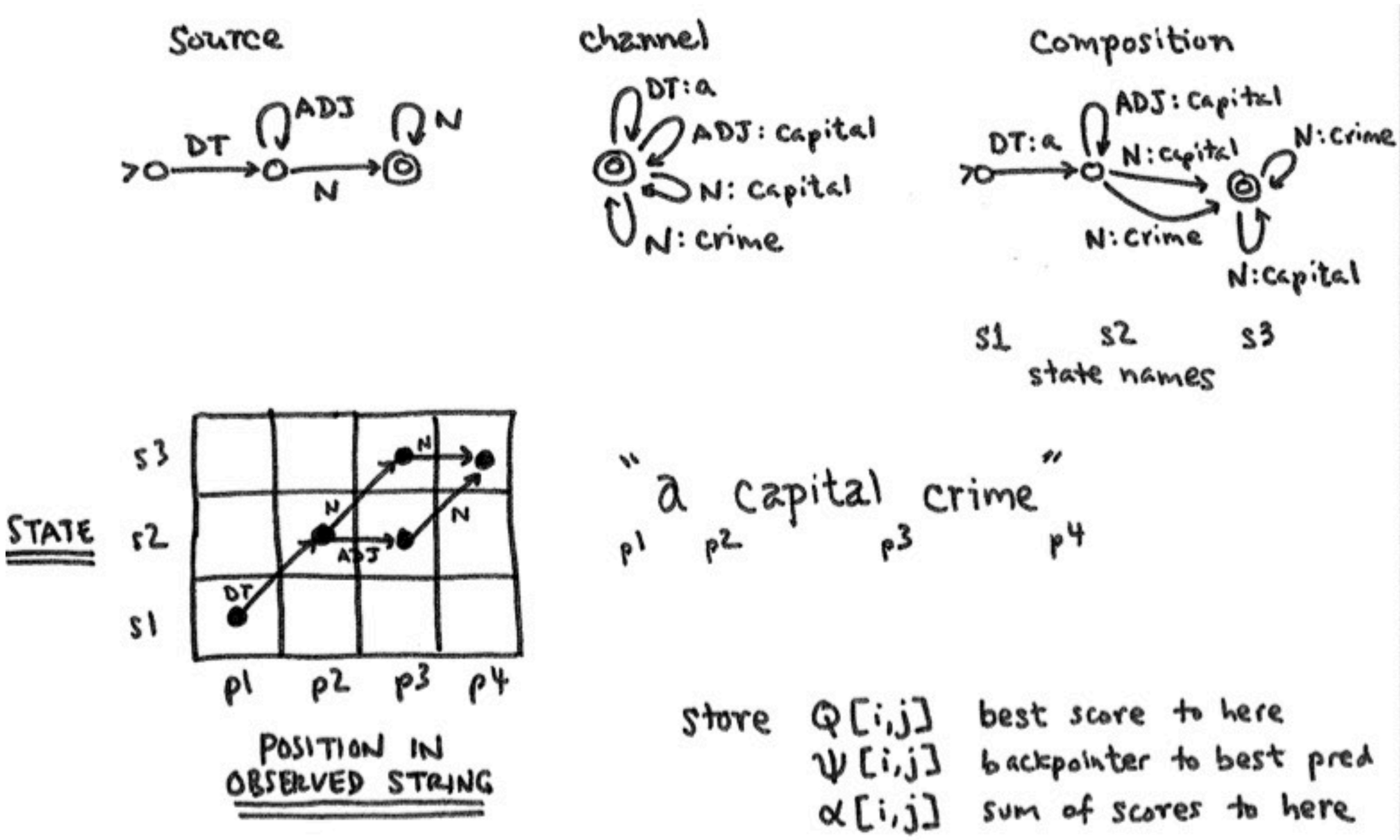
.o.

caca

=

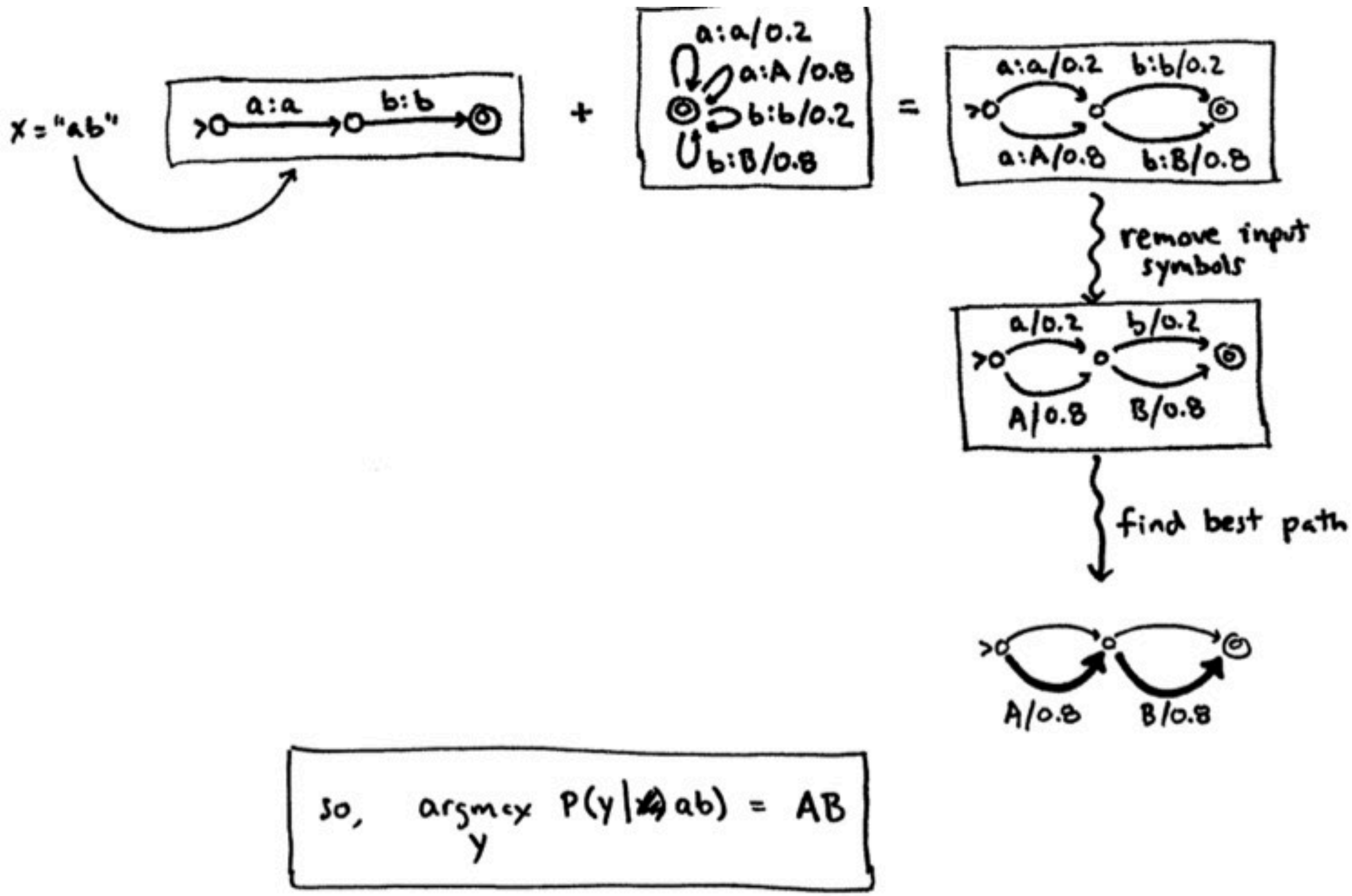


Example: General Tagging

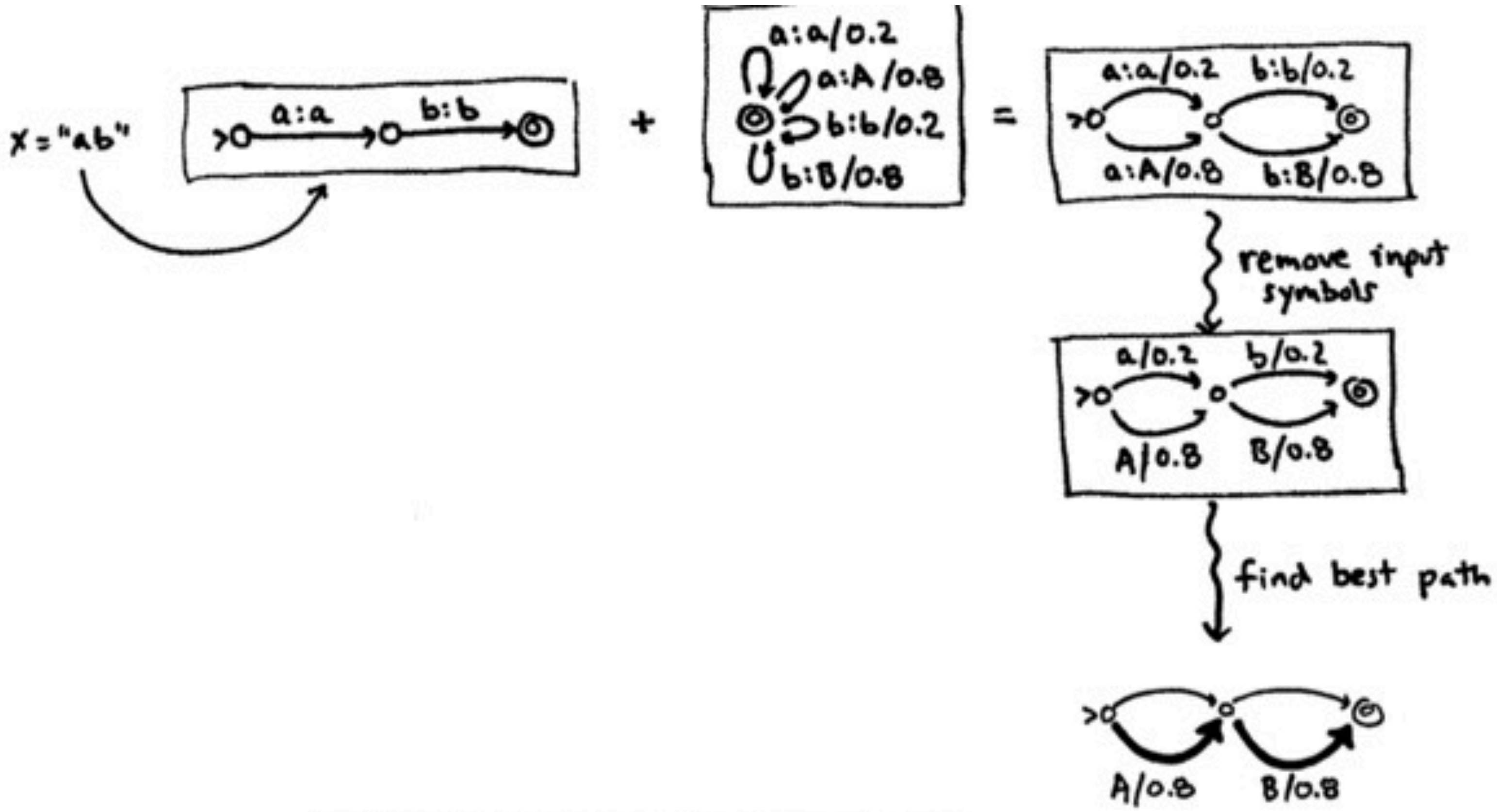


Most Likely "Corrupted Output"

- c) given correct English x, what's the corrupted y with the highest score?

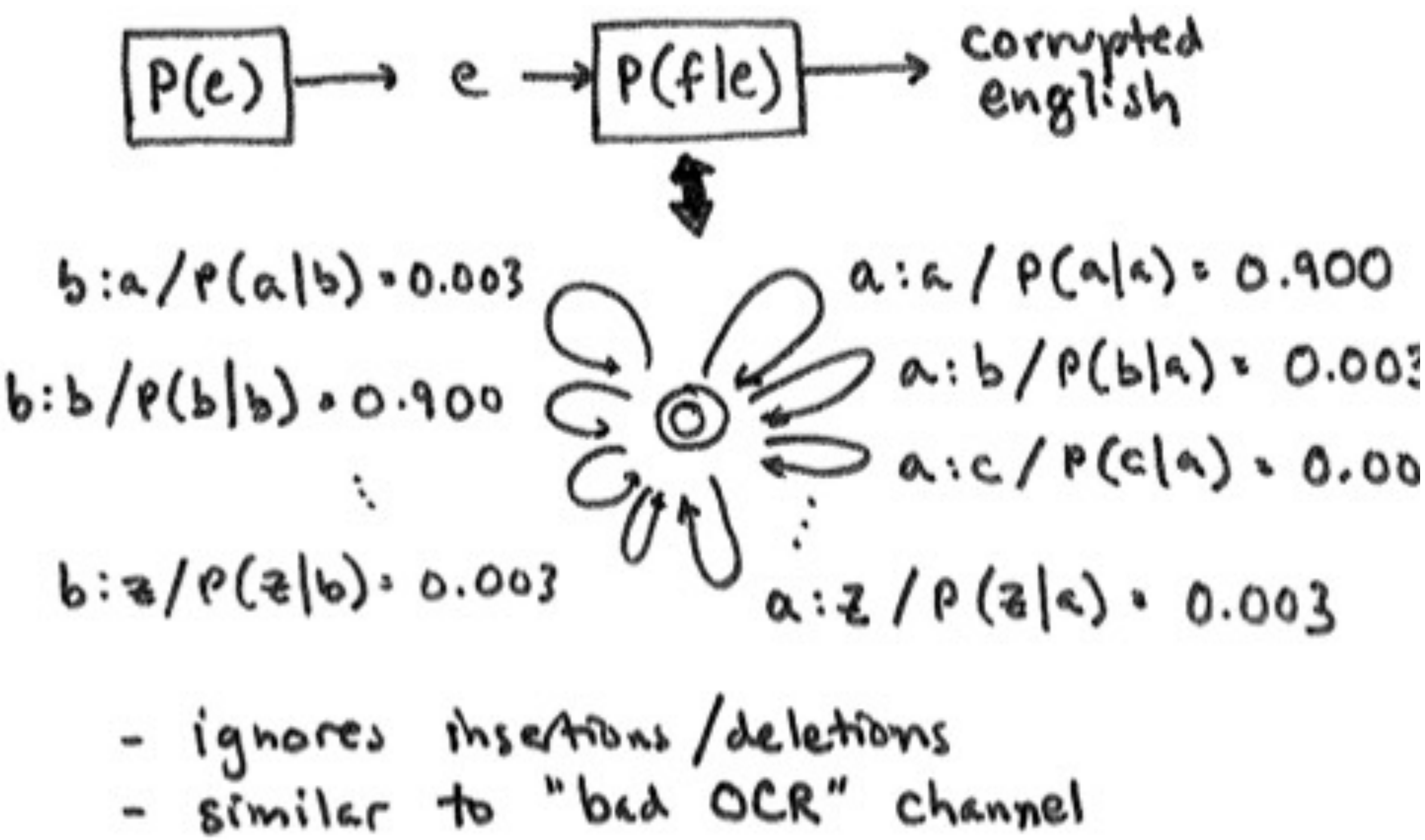


DP for “most likely corrupted”



so, $\arg\max_{y} P(y|ab) = AB$

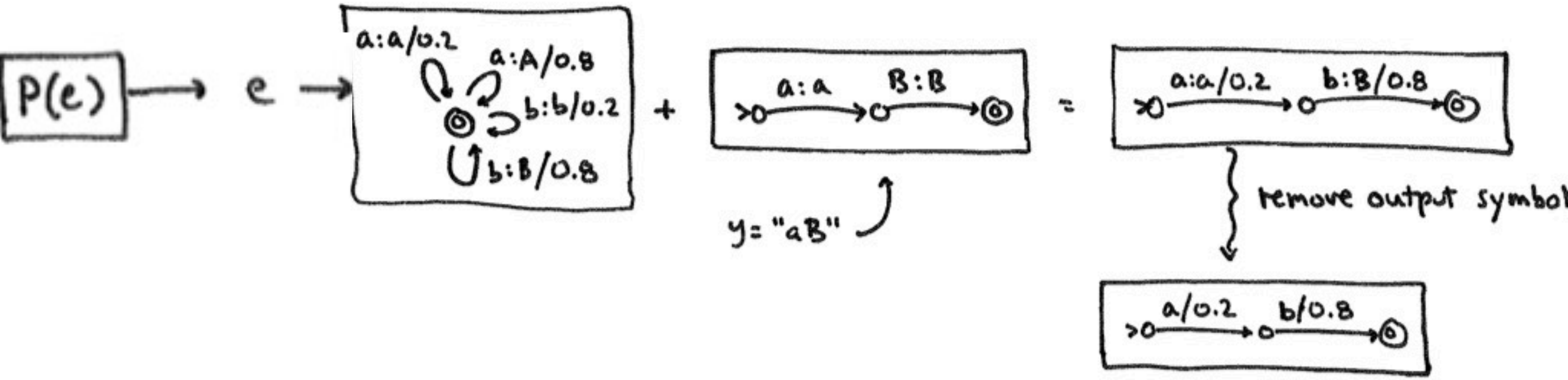
d) Most Likely "Original Input"



- using an LM p(e) as source model for *spelling correction*
  - case 1: letter-based language model $p_L(e)$
  - case 2: word-based language model $p_W(e)$
- How would dynamic programming work for cases 1/2?

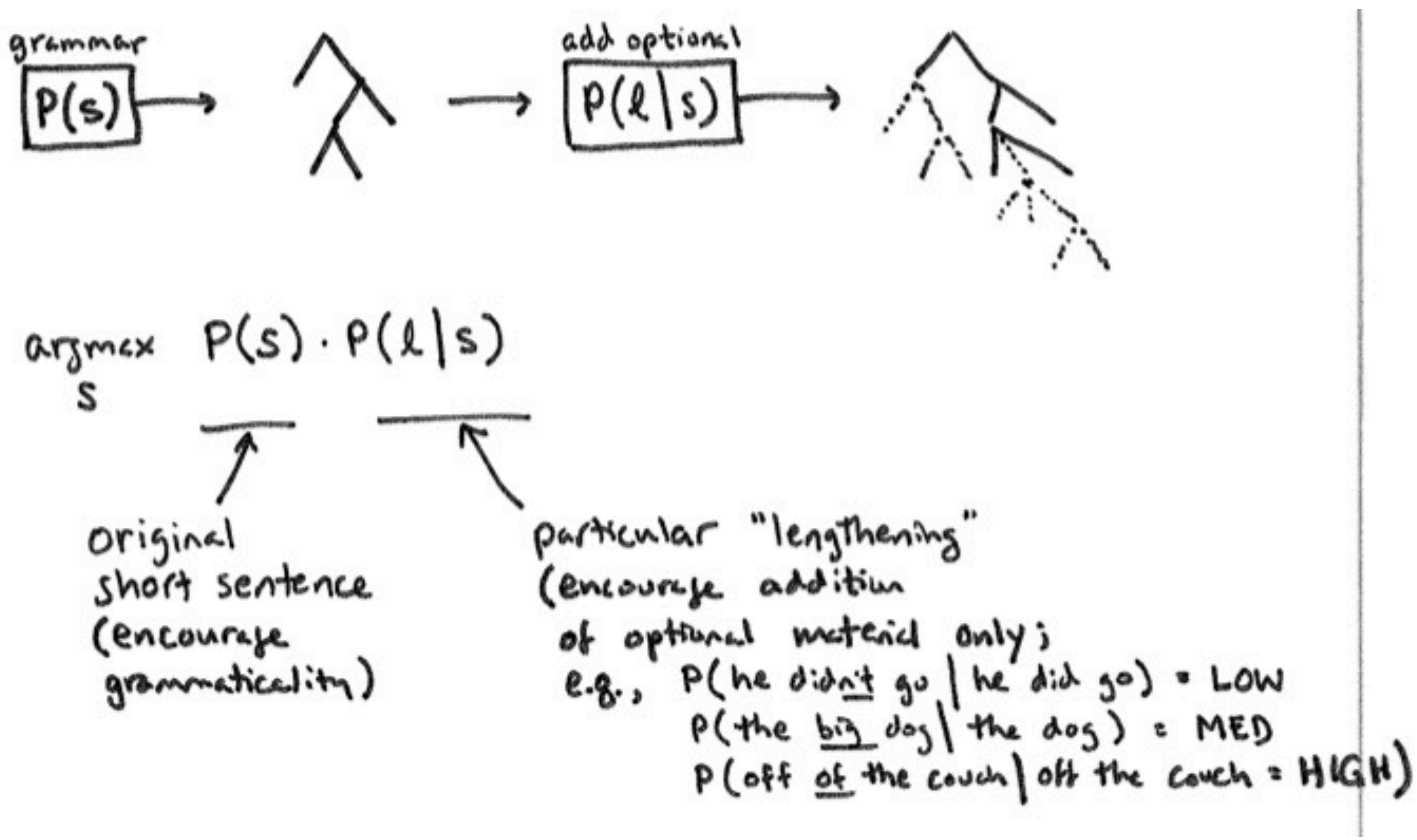
Dynamic Programming for d)

- given y, what is the most likely x with max p(x) p(y|x)



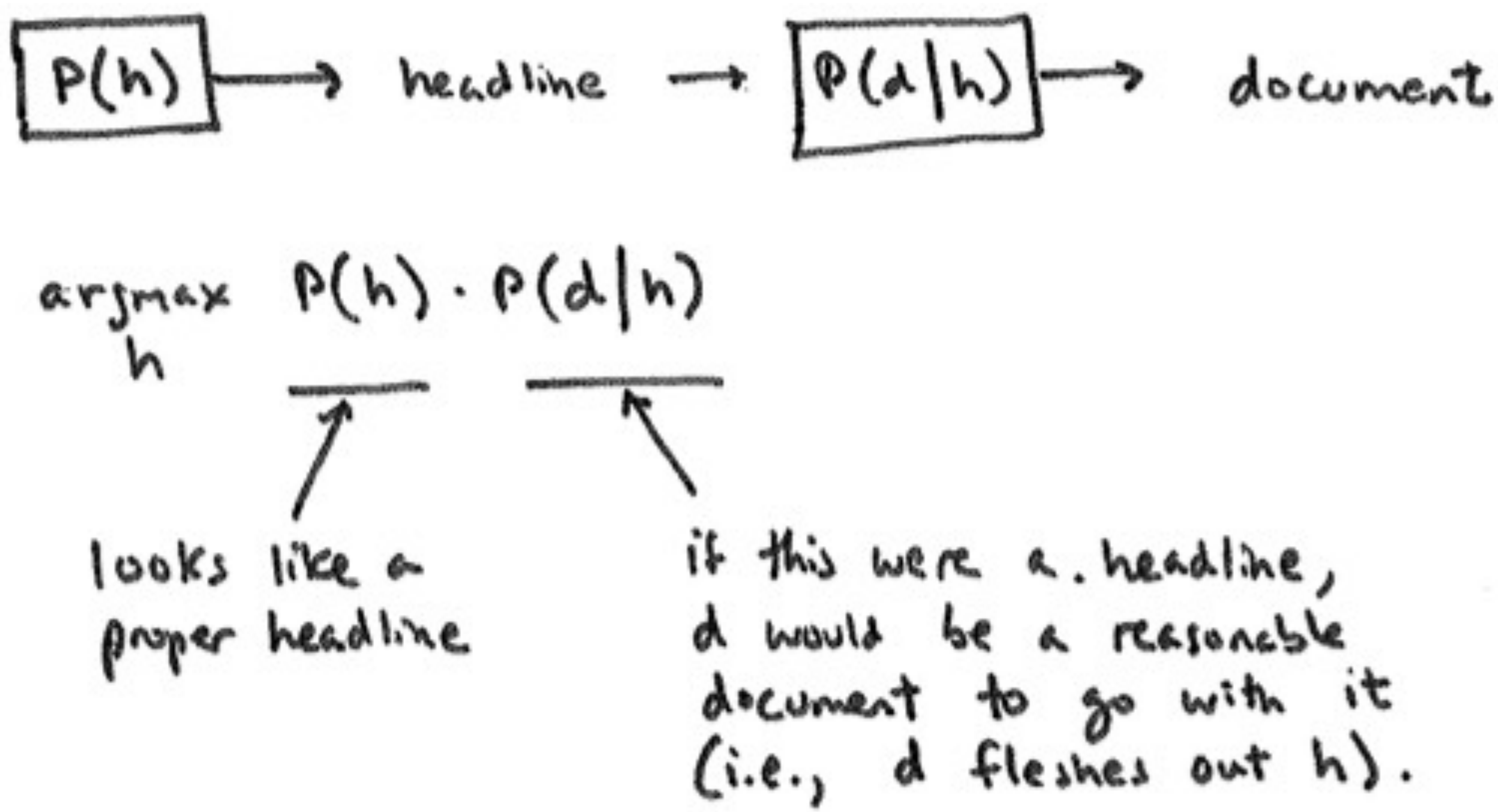
Beyond Finite-State Models

- sentence summarization



Beyond Finite-State Models

- headline generation



Beyond Finite-State Models

- information retrieval

used to rank documents, not construct new ones!

query may contain words not in document.