Finite State Transducers

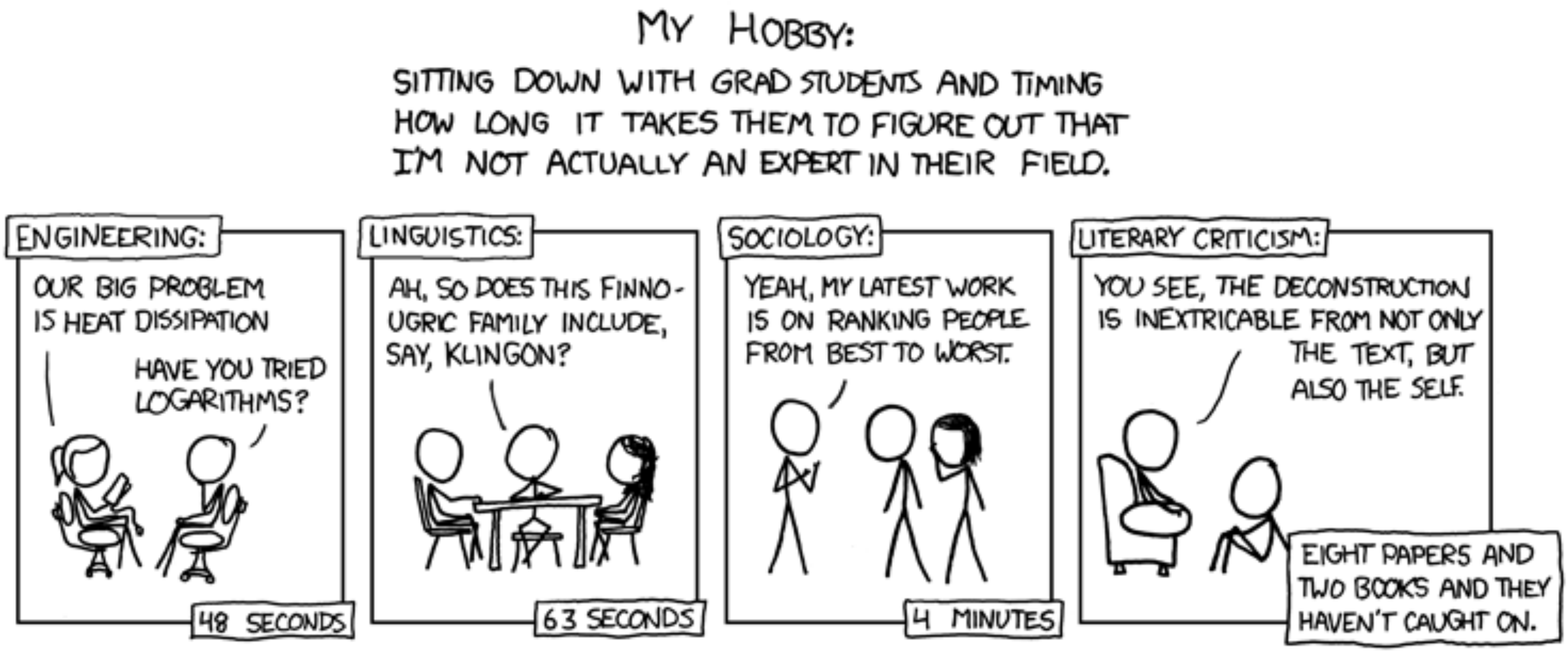
COSI 114 – Computational Linguistics

James Pustejovsky

January 16, 2015

Brandeis University

How to do Linguistics



What is a Finite State Transducer?

- A finite state machine with two tapes: an input tape and an output tape.
- This contrasts with an ordinary finite state automaton (or finite state acceptor), which has a single tape.
- But …
  - How do FSAs and FSTs fit into the larger computational landscape?

Theory of Computation: A Historical Perspective

1930s	• Alan Turing studies Turing machines • Decidability • Halting problem
1940-1950s	• "Finite automata" machines studied • Noam Chomsky proposes the "Chomsky Hierarchy" for formal languages
1969	Cook introduces "intractable" problems or "NP-Hard" problems
1970-	Modern computer science: compilers, computational & complexity theory evolve

Languages & Grammars

Or “**words**”

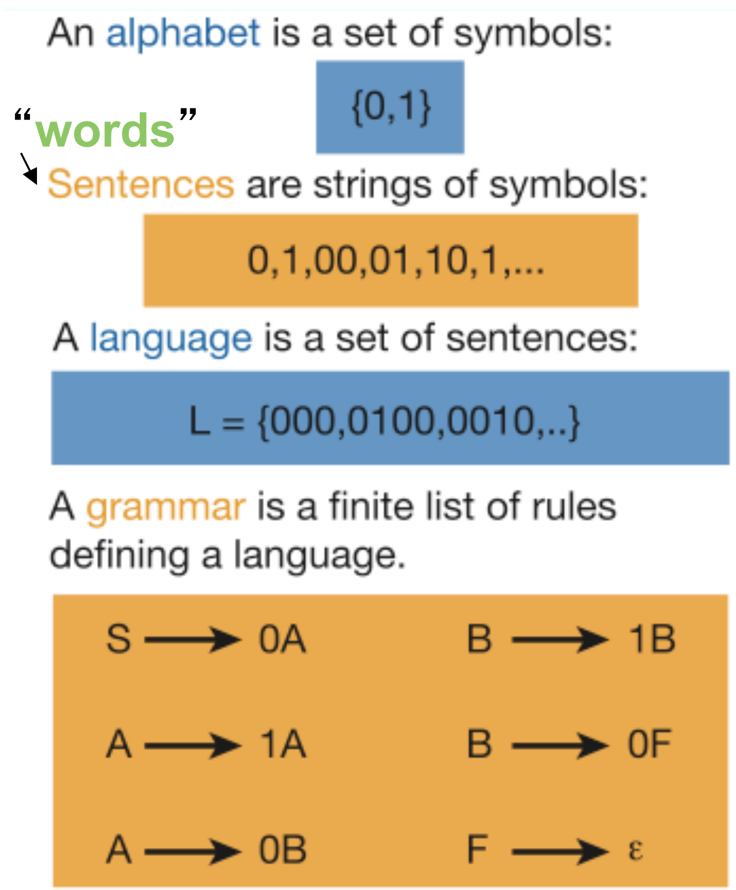
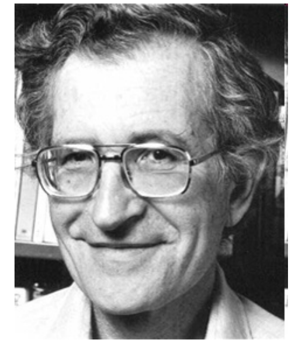


Image source: Nowak et al. Nature, vol 417, 2002

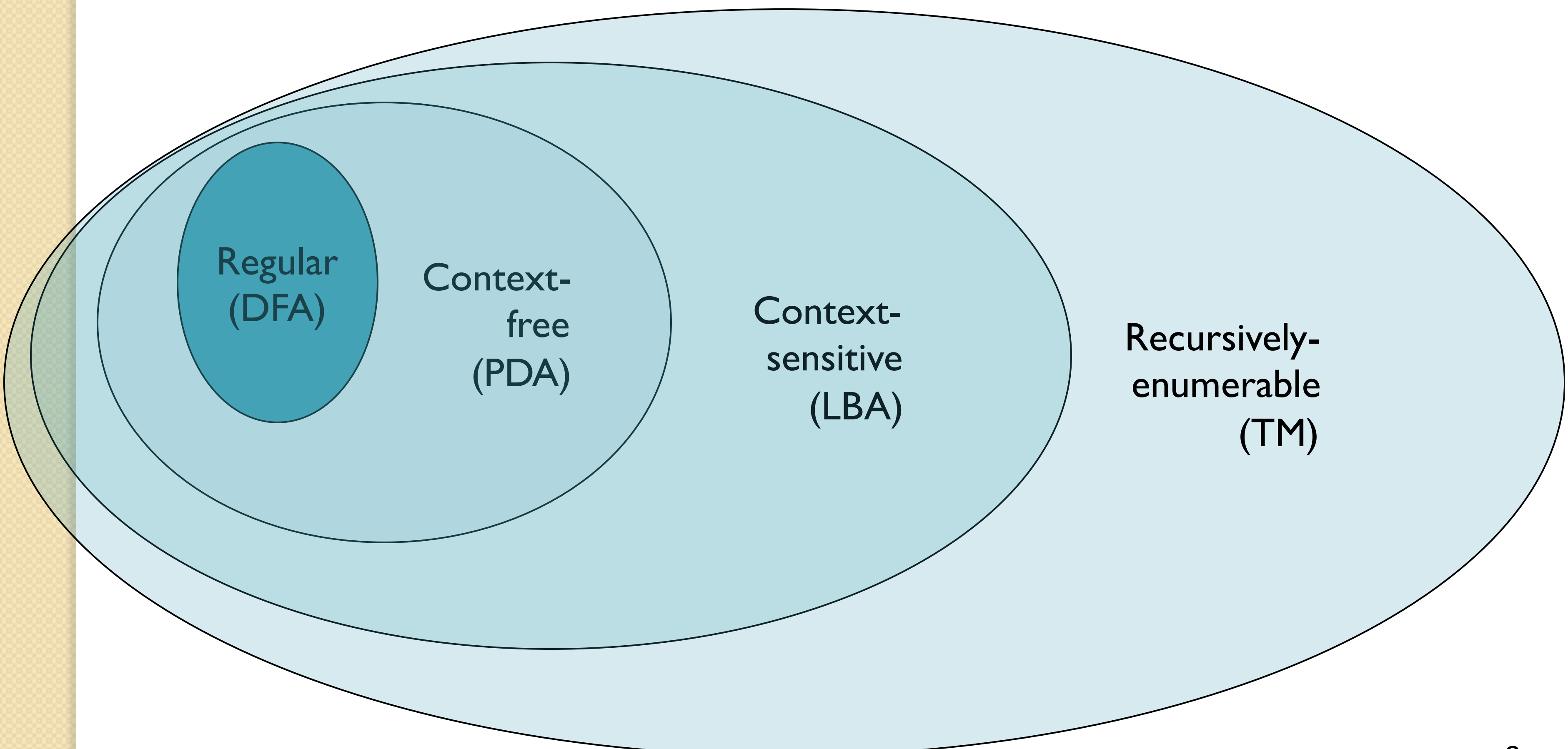
- <u>Languages</u>: *“A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols”*
- <u>Grammars</u>: *“A grammar can be regarded as a device that enumerates the sentences of a language”* - nothing more, nothing less

- *N. Chomsky, Information and Control, Vol 2, 1959*

The Chomsky Hierachy



- A containment hierarchy of classes of formal languages



Alphabet

An alphabet is a finite, non-empty set of symbols

- We use the symbol ∑ (sigma) to denote an alphabet
- Examples:
  - Binary: ∑ = {0,1}
  - All lower case letters: ∑ = {a,b,c,..z}
  - Alphanumeric: ∑ = {a-z, A-Z, 0-9}
  - DNA molecule letters: ∑ = {a,c,g,t}
  - …

Strings

A string or word is a finite sequence of symbols chosen from ∑

- ***Empty string is ε (or "epsilon")***

- Length of a string *w*, denoted by "|*w*|", is equal to the *number of (non- ε) characters in the string*
  - *E.g., x = 010100* *|x| = 6*
  - *x = 01 ε 0 ε 1 ε 00 ε* *|x| = ?*

  - *xy* = concatentation of two strings *x* and *y*

Powers of an alphabet

Let Σ be an alphabet.

- Σ^k = the set of all strings of length *k*
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$

Languages

*L is a said to be a language over alphabet ∑, only if L ⊆ ∑**

➔ this is because ∑* is the set of all strings (of all possible length including 0) over the given alphabet ∑

Examples:

1. Let L be *the* language of all strings consisting of *n* 0's followed by *n* 1's:
   L = {ε,01,0011,000111,…}
2. Let L be *the* language of all strings of with equal number of 0's and 1's:
   L = {ε,01,10,0011,1100,0101,1010,1001,…}

Definition: Ø denotes the Empty language

- Let L = {ε}; Is L=Ø?

NO

The Membership Problem

Given a string $w \in \Sigma^*$ and a language L over Σ, decide whether or not $w \in L$.

Example:

Let w = 100011

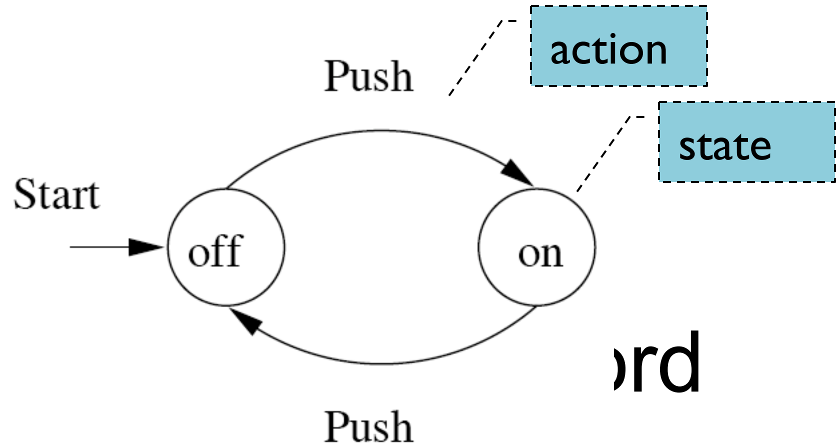
Q) Is $w \in$ the language of strings with equal number of 0s and 1s?

Finite Automata

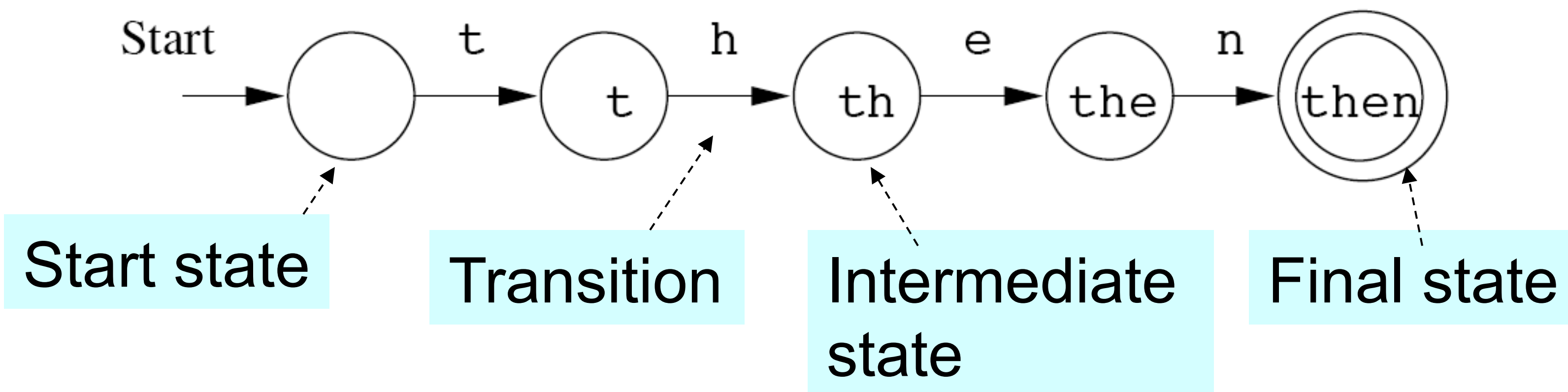
- Some Applications
  - Software for designing and checking the behavior of digital circuits
  - Lexical analyzer of a typical compiler
  - Software for scanning large bodies of text (e.g., web pages) for pattern finding
  - Software for verifying systems of all types that have a finite number of states (e.g., stock market transaction, communication/network protocol)

Finite Automata : Examples

- On/Off switch

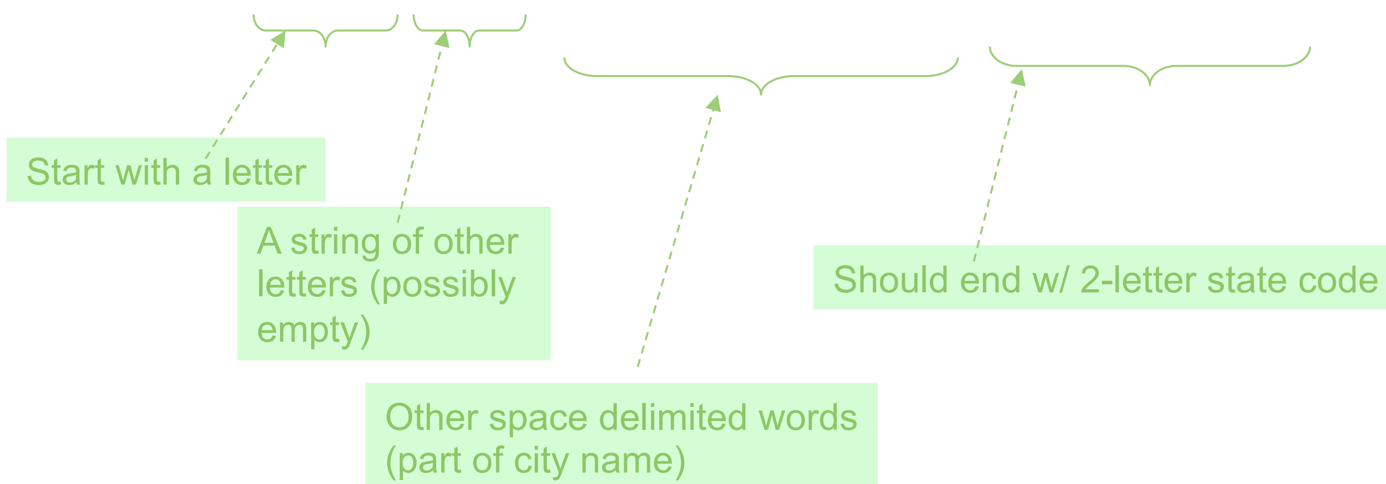


- Modeling recc ord
  "*then*"



Structural expressions

- Grammars
- Regular expressions
  - E.g., unix style to capture city names such as "Palo Alto CA":
    - [A-Z][a-z]*([ ][A-Z][a-z]*)*[ ][A-Z][A-Z]



Some things you can do with FSTs

- **Morphological analysis**
- **Text analysis/normalization**
  - Word segmentation
  - Abbreviation expansion
  - Digit-to-number-name mappings

  i.e. mapping from *writing* to *language*
- Syntactic analysis
  - E.g. part-of-speech tagging
- (With weights) pronunciation modeling and language modeling for speech recognition

What is morphology?

- *scripsērunt* is **third person, plural, perfect, active** of *scrībō* ('I write')
- Morphology relates word forms
  - the "lemma" of *scripsērunt* is *scrībō*
- Morphology analyzes the structure of word forms
  - *scripsērunt* has the structure *scrīb*+s+*ērunt*

Morphology is a relation

- Imagine you have a Latin morphological analyzer comprising:
  - ***D***: a relation that maps between surface form and decomposed form
  - ***L***: a relation that maps between decomposed form and lemma
- Then:
  - *scripsērunt* $\circ$ **D** = *scrīb+s+ērunt*
  - *Scripsērunt* $\circ$ **D** $\circ$ **L** = *scrībō*

English regular plurals

- *cat* + s = *cats* /s/
- *dog* + s = *dogs* /z/
- *spouse* + s = *spouses* /əz/
- This can be implemented by a rule that composes with the base word, inserting the relevant form of the affix at the end

Templatic affixes in Yowlumne

	Neutral Affixes		Template Affixes	
Root	*-al* 'dubitative'	*-t* 'passive aorist'	*-inay* 'gerundial' **CVC(C)**	*-ʃaa* 'durative' **CVCVV(C)**
caw 'shout'	caw-al	caw-t	caw-inay	cawaa-ʃaa-n
cuum 'destroy'	cuum-al	cuum-t	cum-inay	cumuu-ʃaa-n
hoyoo 'name'	hoyoo-al	hoyoo-t	hoy-inay	hoyoo-ʃaa-n
diiyl 'guard'	diiyl-al	diiyl-t	diyl-inay	diyiil-ʃaa-n
ʃilk 'sing'	ʃilk-al	ʃilk-t	ʃilk-inay	ʃiliik-ʃaa-n
hiwiit 'walk'	hiwiit-al	hiwiit-t	hiwt-inay	hiwiit-ʃaa-n

Transducer for each affix transforms base into required templatic form and appends the relevant string.

Subtractive morphology

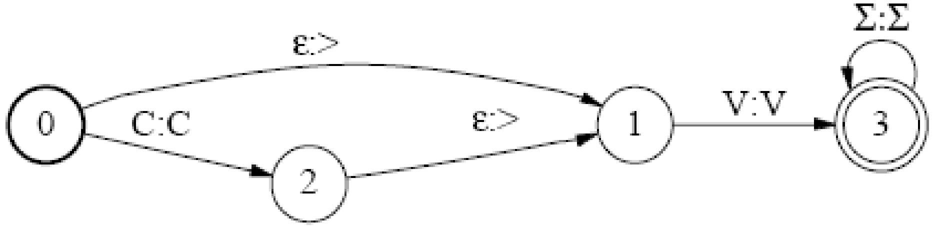
Singular	Plural	Gloss
pitáf-fi-n	**pít**-li-n	'to slice up the middle'
/pitáf-li-n/		
latáf-ka-n	**lat**-ka-n	'to kick something'
tiwáp-li-n	**tiw**-wi-n	'to open something'
/tiw-li-n/		
atakáa-li-n	**aták**-li-n	'to hang something'
icoktakáa-li-n	**icokták**-li-n	'to open one's mouth'
albitíi-li-n	**albít**-li-n	'to place on top of'
ciłíp-ka-n	**cíł**-ka-n	'to spear something'
facóo-ka-n	**fas**-ka-n	'to flake off'
/fac-ka-n/		
onasanáy-li-n	**onasan**-níici-n	'to twist something on'
iyyakohóp-ka-n	**iyyakóf**-ka-n	'to trip'
/iyyakóh-ka-n/		
koyóf-fi-n	**kóy**-li-n	'to cut something'
/koyóf-li-n/		

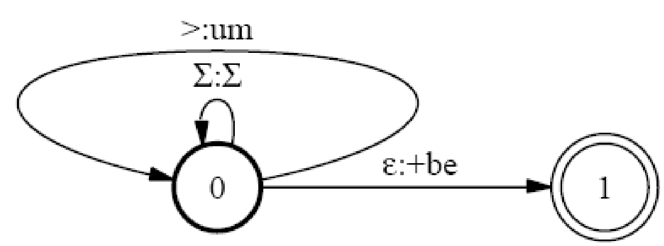
Transducer deletes final VC of the base…

Bontoc infixation

antj´ŏak	‘tall’	*umantj´ŏak*	‘I am getting taller’
k´ăwĭsat	‘good’	*kum´ăwĭsat*	‘I am getting better’
p´ūsiak	‘poor’	*pum´ūsiak*	‘I am getting poorer’

- Insert a marker “>” after the first consonant (if any)
- Change “>” into the infix *–um-*





Reduplication: Gothic

Infinitive	Gloss	Preterite
falþan	'fold'	faífalþ
haldan	'hold'	haíhald
ga-staldan	'possess'	ga-staístald
af-áikan	'deny'	af-aíáik
máitan	'cut'	maímáit
skáidan	'divide'	skaískáiþ
slēpan	'sleep'	saíslēp
grētan	'greet'	gaígrōt
ga-rēdan	'reflect upon'	ga-raírōþ
tēkan	'touch'	taítōk
saian	'sow'	saísō

Problem: mapping *w* to *ww* is not a regular relation

Factoring Reduplication

- Prosodic constraints

$$\alpha = \beta \circ R = (A_1)C_2 a\acute{\imath}\beta' \qquad X_1 X_2 a\acute{\imath} s_1 \dot{k}_2 \acute{a} i \text{þ}$$

- Copy verification transducer *C*

$$\bigcup_{s \in segments} \neg[\Sigma^* s_1 \Sigma^* \neg s_1 \Sigma^*] \qquad \bigcup_{i \in indices} \bigcup_{s \in segments} \neg[\Sigma^* s_i \Sigma^* \neg s_i \Sigma^*]$$

Non-Exact Copies

- Dakota (Inkelas & Zoll, 1999):

kičaxčaγa ‘he made it for them quickly’

Non-Exact Copies

- Basic and modified stems in Sye (Inkelas & Zoll, 1999):

cw-amol-omol

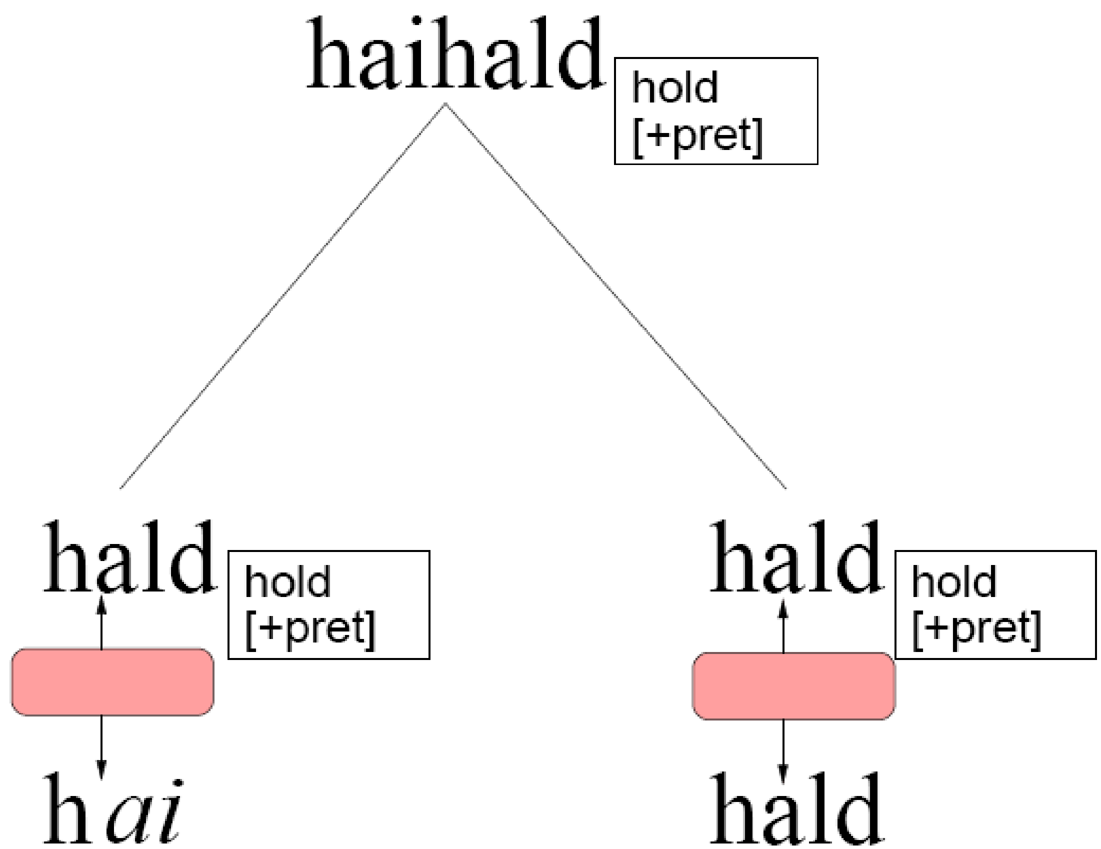
"they will fall all over"

Basic	Modified	Gloss
evcah	ampcah	'defecate'
evinte	avinte	'look after'
evsor	amsor	'wake up'
evtit	avtit	'meet'
ocep	agkep	'fly'
ochi	aghi	'see it'
omol	amol	'fall'
oruc	anduc	'bathe'
ovoli	ampoli	'turn it'
ovyu-	avyu-	(causative prefix)
owi	awi	'leave'
pat	ampat	'blocked'
vag	ampag	'eat'

Morphological Doubling Theory
(Inkelas & Zoll, 1999)

- Most linguistic accounts of reduplication assume that the copying is done as part of morphology
- In MDT:
  - Reduplication involves doubling at the *morphosyntactic* level – i.e. one is actually simply repeating words or morphemes
  - Phonological doubling is thus expected, but not required

Gothic Reduplication under Morphological Doubling Theory



Another Example: Linguistic analysis of text

- Maps between the stuff you see on the page – e.g. text written in the standard orthography of a language – into linguistic units (words, morphemes, phonemes…)
- For example:
  - I ate a 25kg bass
  - [aɪ ɛɪt ə twɛnti faɪv kɪləgræm bæs]
- This can be done using transducers
  - But is the mapping between writing and language *really* regular (finite-state)?

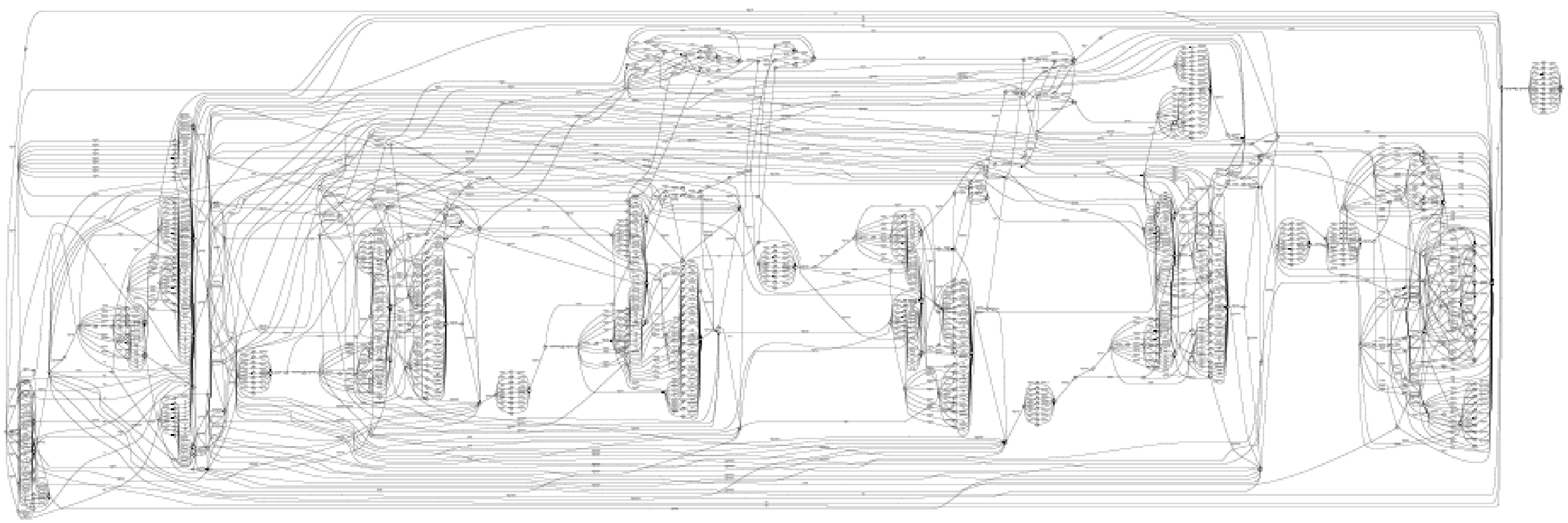
Linguistic analysis of text

- Abbreviation expansion
- Disambiguation
- Number expansion
- Morphological analysis of words
- Word pronunciation
- …

A transducer for number names

Consider a machine that maps between digit strings and their reading as number names in English.

30,294,005,179,018,903.56 →
thirty quadrillion, two hundred and ninety four trillion, five billion, one hundred seventy nine million, eighteen thousand, nine hundred three, point five six

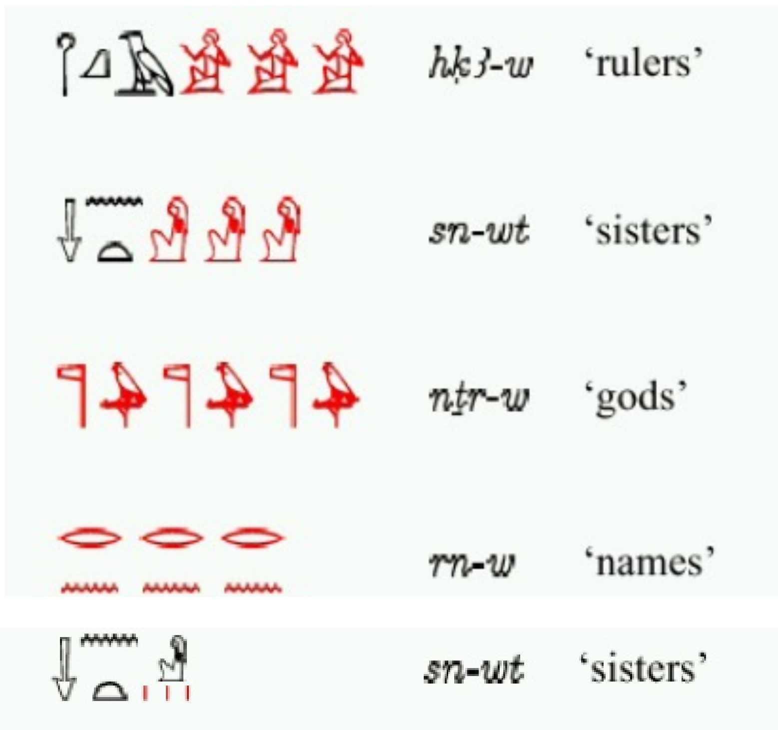


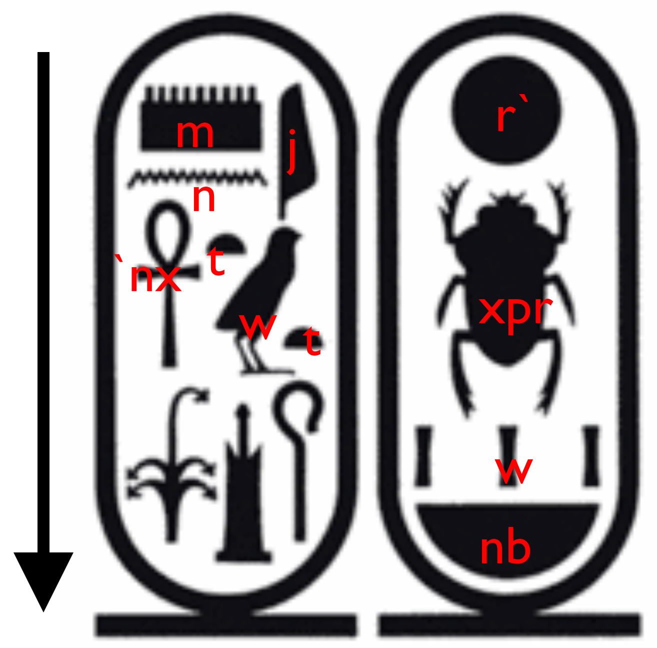
Mapping between speech and writing

It seems obvious on the face of it that the mapping between speech and its written form is regular. After all, the words are ordered in the same way as speech. Even the letters tend to be ordered in the same way as the sounds they represent.

Some examples where it isn't…

'honorific inversion'





Finite state methods

- In morphology they seem almost exactly correct as characterizations of the natural phenomenon

- In the mapping from writing to language, again, finite-state models seem almost exactly correct

Text Normalization

- Conversion of text that includes 'non-standard' words like numbers, abbreviations, misspellings . . . into normal words.
  - Abbreviation expansion (including novel abbreviations)
  - Expansion of numbers into 'number names'
  - Correction of misspellings
  - Disambiguation in cases where there is ambiguity

Where is normalization needed?

- Very little in cases like this:

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

So she was considering in her own mind (as well as she could, for the hot day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

Where is normalization needed?

- A lot in cases like this:

CUST RCVD LTTR CNCRNG LOCAL SRVC

VISIT NECESSARY BUT CST STILL HAS PAC BELL SERV ON OLD TN AT RESIDENCE

ORDERD CALLNG CRDS PER CSR RQST

1st att, left mssg for CB from Lynda, will wait for call

50's Sutton Place Area Convertible 3BR 1400 SF 2BR, 2Bth, L-Shaped LR, S.E. Open Vus, Gar, Rf Dk, Mid $400K's Thompson Kane Ina 339-8300

57 ST E/1st & 2nd Ave Huge drmn 1 BR 750+ sf, lots of sun & clsts. Sundeck & lndry facils. Askg $187K, maint $868, utils incld. Call Bkr Peter 914-428-9054.

Humans are pretty good at this: can you read this?

f u cn rd ths thn u r dng btr thn ny autmtc txt nrmlztion prgrm cn do.

How about this?

Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in what oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae. The rset can be a total mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe.

Or this?

Goccdrnia to a hscheearcr at Emabrigdc Yinervtisu, it teosn'd rttaem in tahw rredo the stteerl in a drow are, the ylno tprmoetni gihnt is taht the trisf and tsal rtteel be at the tghir eclap. The tser can be a lotat ssem and you can litls daer it touthiw morbelp. Siht is ecuseab the nuamh dnim seod not daer yrvee rtetel by fstlei, but the drow as a elohw.

Two components of text normalization

- Given a string of characters in a text, what is the (reasonable) set of possible actual words (or word sequences) that might correspond to it.
- Which of those is right for the particular context?

An illustration

Hlivetast 123 kfiangsAvienue.s

Two components of text normalization

- A component that gives you the set of possibilities:
  - *123 = one hundred (and) twenty three*
  - *123 = one twenty three*
  - *123 = one two three*
- A component that tells you which one(s) are appropriate to a particular context.

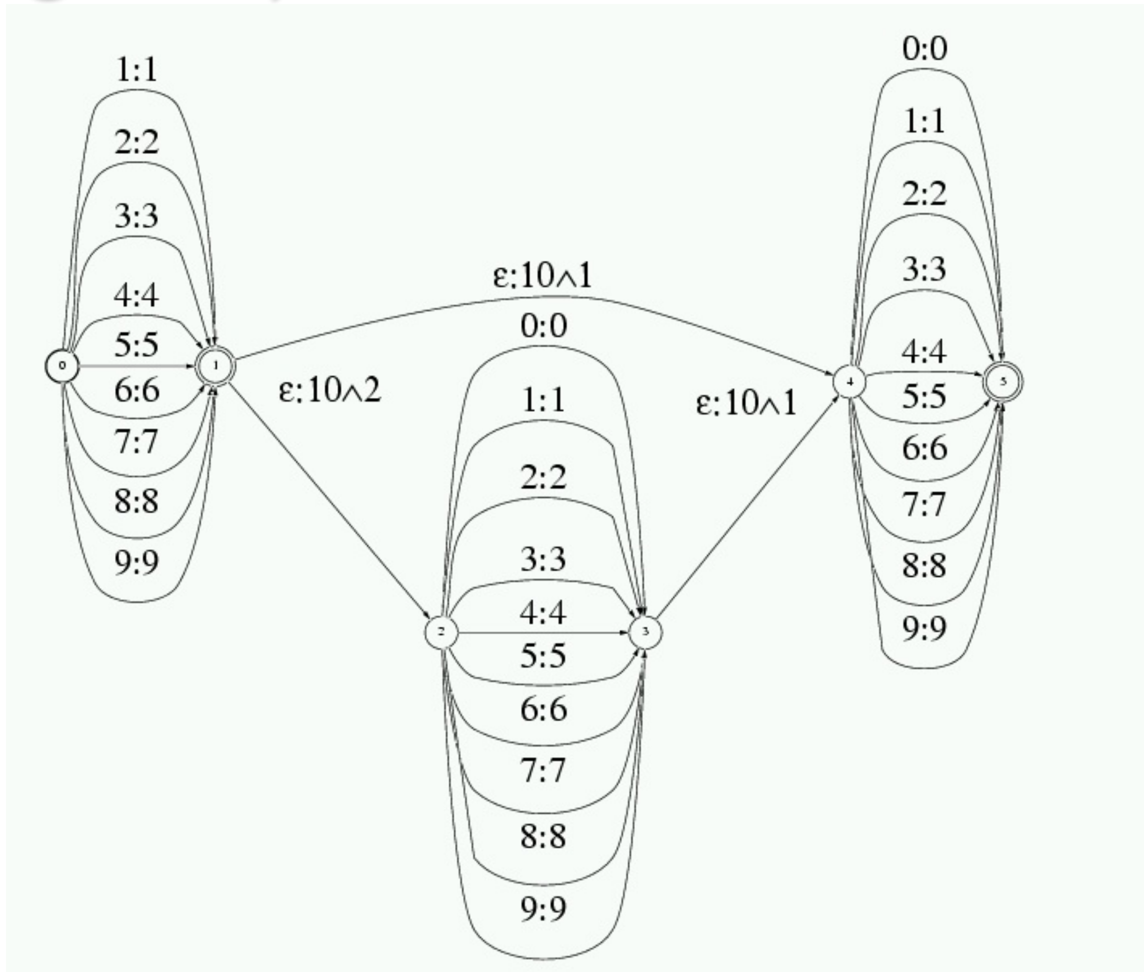
A concrete example of finite-state methods in text normalization: digit to number name translation

- Factor digit string:
  - *123* → $\mathbf{1 \cdot 10^2 + 2 \cdot 10^1 + 3}$
- Translate factors into number names:
  - $\mathbf{10^2}$ → *hundred*
  - $\mathbf{2 \cdot 10^1}$ → *twenty*
  - $\mathbf{1 \cdot 10^1 + 3}$ → *thirteen*
- Languages vary on how extensive these lexicons are. Some (e.g. Chinese) have very regular (hence very simple) number name systems; others (e.g. Urdu/Hindi) have a large set of number names with a name for almost every number from 1 to 100.
- Each of these steps can be accomplished with FSTs

Urdu (Hindi) Number Names

1	eik	21	ik-kees	41	ikta-lees	61	ik-shat	81	ik-si
2	dau	22	ba-ees	42	baya-lees	62	ba-shat	82	baya-si
3	teen	23	ta-ees	43	tainta-lees	63	tere-shat	83	tera-si
4	chaar	24	chau-bees	44	chawa-lees	64	chaun-shat	84	chaura-si
5	paanch	25	pach-chees	45	painta-lees	65	paen-shat	85	picha-si
6	chay	26	chab-bees	46	chaya-lees	66	sar-shat / chay-aa-shat	86	chaya-si
7	saath	27	satta-ees	47	santa-lees	67	sataath	87	sata-si
8	aath	28	attha-ees	48	arta-lees	68	athath	88	atha-si
9	nau	29	unat-tees	49	un-chas	69	unat-tar	89	
10	dus	30	tees	50	pa-chas	70	sat-tar	90	navay
11	gyaa-raan	31	ikat-tees	51	ika-vun	71	ikat-tar	91	ikan-vay
12	baa-raan	32	bat-tees	52	ba-vun	72	bahat-tar	92	ban-vay
13	te-raan	33	tain-tees	53	tera-pun	73	tehat-tar	93	teran-vay
14	chau-daan	34	chaun-tees	54	chav-van	74	chohat-tar	94	chauran-vay
15	pand-raan	35	pan-tees	55	pach-pan	75	pagat-tar	95	pichan-vay
16	so-laan	36	chat-tees	56	chap-pan	76	chayat-tar	96	chiyan-vay
17	sat-raan	37	san-tees	57	sata-van	77	satat-tar	97	chatan-vay
18	attha-raan	38	ear-tees	58	atha-van	78	athat-tar	98	athan-vay
19	un-nees	39	unta-lees	59	un-shat	79	una-si	99	ninan-vay
20	bees	40	cha-lees	60	shaat	80	assi	100	saw

Digit string factoring transducer (fragment)



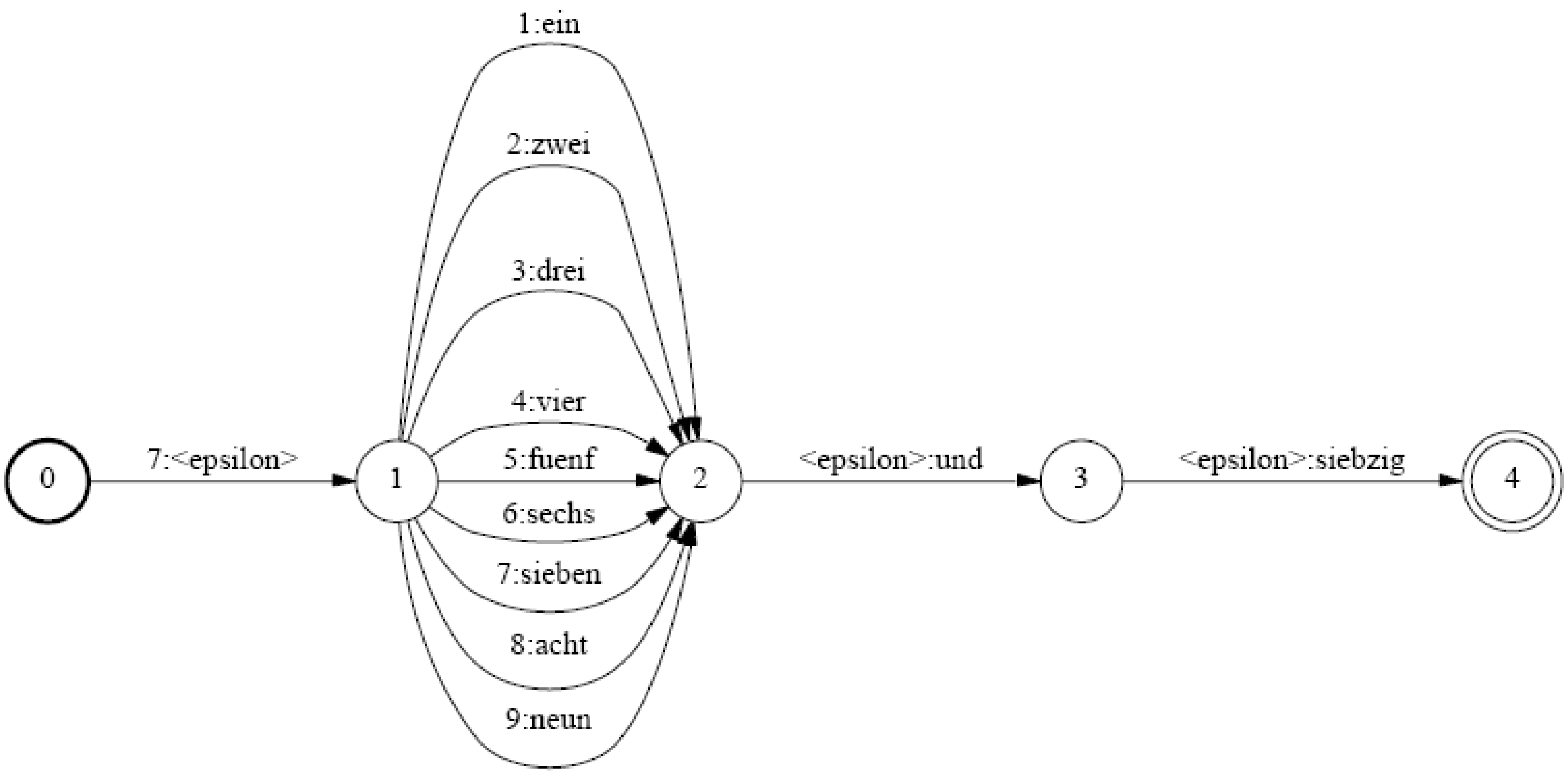
Germanic "decade flop"

zwanzig vier

24

und

70’s


0
7:<epsilon>
1
1:ein
2:zwei
3:drei
4:vier
5:fuenf
6:sechs
7:sieben
8:acht
9:neun
2
<epsilon>:und
3
<epsilon>:siebzig
4

Digit-string to number name translation: German

- Factor digit string:
  - *123* $\rightarrow 1 \cdot 10^2 + 2 \cdot 10^1 + 3$
- **Flip decades and units:**
  $2 \cdot 10^1 + 3 \rightarrow 3 + 2 \cdot 10^1$
- Translate factors into number names:
  - $10^2 \rightarrow$ *hundert*
  - $2 \cdot 10^1 \rightarrow$ *zwanzig*
  - $1 \cdot 10^1 + 3 \rightarrow$ *dreizehn*

German number grammar (fragment)

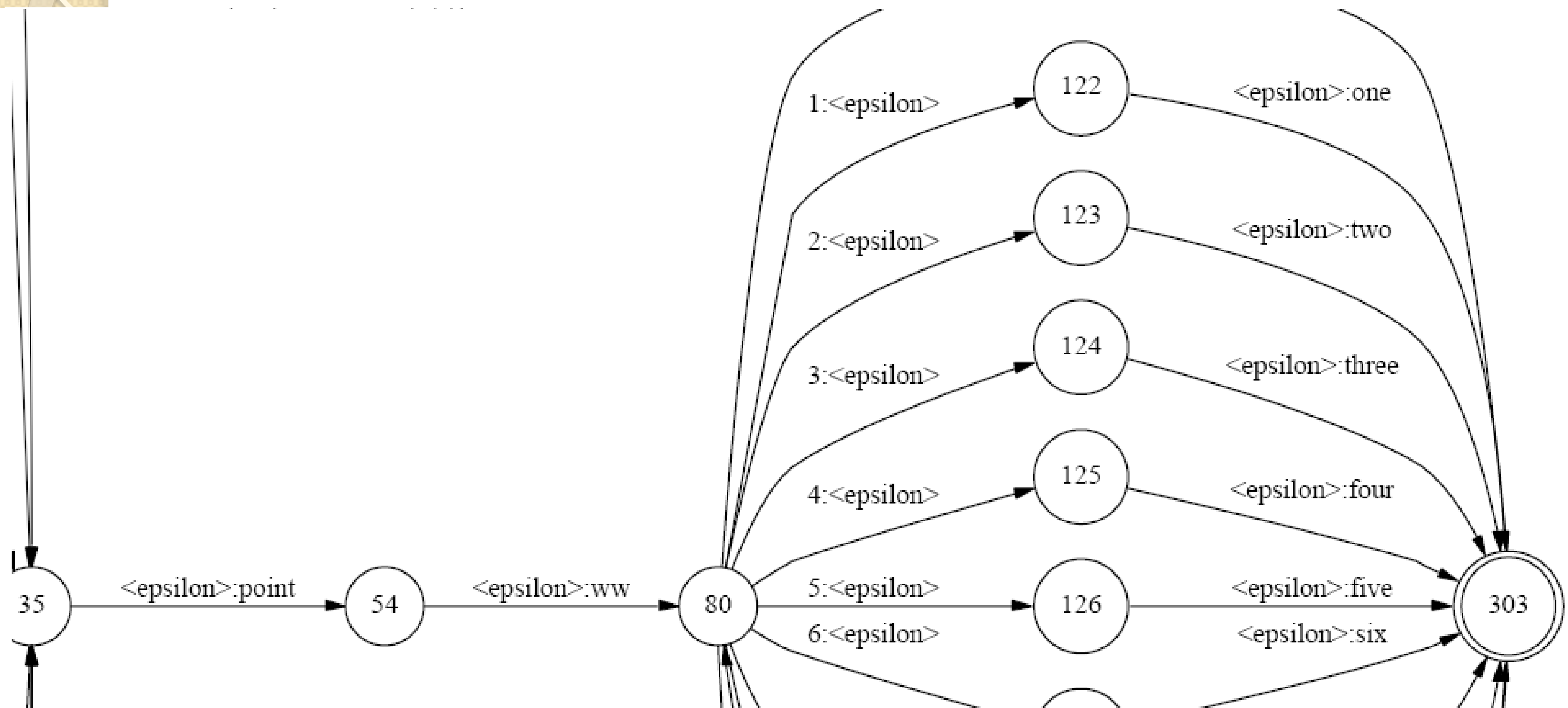
$$
\begin{array}{lcl}
\text{TEN} & \rightarrow & 1 \cdot 10^1 \text{ zehn} \mid \text{TEENW} \\
\text{TEN} & \rightarrow & \text{UNITW und TENW} \\
\text{TEN} & \rightarrow & \text{UNITW} \\
\text{TEN} & \rightarrow & \text{TENW} \\
 & & \\
\text{TENW} & \rightarrow & 2 \cdot 10^1 \text{ zwanzig} \mid \\
 & & 3 \cdot 10^1 \text{ dreißig} \mid \\
 & & 4 \cdot 10^1 \text{ vierzig} \mid \\
 & & 5 \cdot 10^1 \text{ fünfzig} \ldots \\
 & & \\
\text{TEENW} & \rightarrow & 1 \cdot 10^1 + 1 \text{ elf} \mid \\
 & & 1 \cdot 10^1 + 2 \text{ zwölf} \mid \\
 & & 1 \cdot 10^1 + 3 \text{ dreizehn} \mid \\
 & & 1 \cdot 10^1 + 4 \text{ vierzehn} \ldots
\end{array}
$$

Concrete example from English

Consider a machine that maps between digit strings and their reading as number names in English.

30,294,005,179,018,903.56 →
thirty quadrillion, two hundred and ninety four trillion, five billion, one hundred seventy nine million, eighteen thousand, nine hundred three, point five six

566 states and 1492 arcs



The Problem: Rampant Abbreviations

- UNE-P RAMP notes:
  CUST RCVD LTTR CNCRNG LOCAL SRVC
  VISIT NECESSARY BUT CST STILL HAS PAC BELL SERV ON OLD TN AT RESIDENCE
  ORDERD CALLNG CRDS PER CSR RQST

- Worldnet notes:
  Cust wanted to know if we currently had 4.95 pp Adv we do not
  cust still has at&t s/w on comp he is going to be moving to PA in a mth and wants to know if he can reactivate this acct

- LIFE Remarks:
  1st att, left mssg for CB from Lynda, will wait for call
  CUST REQUESTD CHANGE IN HUNTING, FOLLOW ORDER. NO CSR FOUND. CUST WITH RESELLER ALEGIANCE.

What do I mean by "Abbreviation"?

Any word that is shortened from its normal spelling, but that should be read as if it were spelled in full.

Under this definition:

- *cust* and *mth* are abbreviations since they are clearly to be read *customer*, *month*
- *NATO*, *UN*, CSR are not abbreviations since they are standardly read as words ("acronyms") or sequences of letters.
- Some terms such as *LD* ("long distance") have become pretty standard, and so will not be treated as abbreviations.

NSW Classification

TABLE I. Taxonomy of non-standard words used in hand-tagging and in the text normalization models

	EXPN	abbreviation	*adv, N.Y, mph, gov't*
alpha	LSEQ	letter sequence	*CIA, D.C, CDs*
	ASWD	read as word	*CAT*, proper names
	MSPL	misspelling	*geogaphy*
	NUM	number (cardinal)	*12, 45, 1/2, 0·6*
	NORD	number (ordinal)	*May 7, 3rd, Bill Gates III*
	NTEL	telephone (or part of)	*212 555-4523*
	NDIG	number as digits	*Room 101*
N	NIDE	identifier	*747, 386, I5, pc110, 3A*
U	NADDR	number as street address	*5000 Pennsylvania, 4523 Forbes*
M	NZIP	zip code or PO Box	*91020*
B	NTIME	a (compound) time	*3·20, 11:45*
E	NDATE	a (compound) date	*2/2/99, 14/03/87* (or US) *03/14/87*
R	NYER	year(s)	*1998, 80s, 1900s, 2003*
S	MONEY	money (US or other)	*$3·45, HK$300, Y20,000, $200K*
	BMONEY	money tr/m/billions	*$3·45 billion*
	PRCT	percentage	*75%, 3·4%*
	SPLT	mixed or "split"	*WS99, x220, 2-car*
			(see also SLNT and PUNC examples)
	SLNT	not spoken,	word boundary or emphasis character:
M		word boundary	*M.bath, KENT*RLTY, _really_*
I	PUNC	not spoken,	non-standard punctuation: "***" in
S		phrase boundary	*$99,9K***Whites*, "..." in *DECIDE...Year*
C	FNSP	funny spelling	*slloooooww, sh*t*
	URL	url, pathname or email	*http://apj.co.uk, /usr/local, phj@tpt.com*
	NONE	should be ignored	ascii art, formatting junk

Normalization

cci vm not wrking has not fully complted xfer to svc

One Approach

Large script with lots of rules:

- s/ AN ADV / AN ADVERTISEMENT /
  s/ 2 ADVS* / TO ADVISE /
  s/TO ADVS* / TO ADVISE /
  s/ ADVS*D* / ADVISED /g
  s/ AMER[.]* / AMERICA /
  s/ AMT / AMOUNT /

- Cf. U Penn Linguistic Data Consortium's "Text Conditioning Tools"

Problem: How many ways can you spell *customer* in UNE-P RAMP?

1.	cmr dscnnctd	customer disconnected
2.	com upset	customer upset
3.	cs clg	customer calling
4.	csmr cllng	customer calling
5.	csr called	customer called
6.	cst understood	customer understood
7.	cstm wnts	customer wants
8.	cstmr advsd	customer advised
9.	cstr claims	customer claims
10.	csu req	customer request
11.	csut wntd	customer wanted
12.	cts called	customer called
13.	cu called	customer called
14.	cus advised	customer advised
15.	cust care	customer care
16.	custm clld	customer called
17.	custo call	customer call
18.	customer chngd	customer changed
19.	custr upst	customer upset

Corpus-Dependent Unsupervised Abbreviation Expansion (Sproat et al. 2001)

Problem: given a previously unseen abbreviation, how do you use corpus-internal evidence to find the expansion into a *standard word*?

Example:	cus wnt info on services and chrgs
Elsewhere in Corpus:	. . . customer wants . . .
	. . . wants info on vmail . . .

A Source-Channel Language Model Approach

$$\hat{\mathbf{w}} \approx \mathrm{argmax}_{\mathbf{w},\mathbf{t}}\, p(\mathbf{o}|\mathbf{t},\mathbf{w})p(\mathbf{t}|\mathbf{w})p(\mathbf{w})$$

Where:

- **o** are the *observed text*
- **w** are the *underlying words*
- **t** are the *tags* (in this case the tags "abbreviate" and "don't abbreviate")

WFST-based Implementation

$T' = \pi_2(ShortestPath(T \circ A^{-1} \circ L))$

Where:

- T is text
- T' is normalized text
- A is abbreviation model
- L is language model

cf. *CLG* model used in ASR

Processing Steps

- Preprocess text ("splitter").
- Collect possible abbreviations and their possible expansions; use a stoplist of things not to expand.
- Train a language model on "clean" text .
- Normalize text.

Splitter

- ORD#C219*XXXXXXX* V2-REJ 9481 FEA DOES NOT EXIST ON ACCT/2ND ATTEMPT/TO BE PLACED IN TTID GA-CWD/IF CUS CALLS PLEASE REFER TO OM VERIBAGE
- ord # c 219*XXXXXXX* v 2 - rej 9481 fea does not exist on acct / 2nd attempt / to be placed in ttid ga - cwd / if cus calls please refer to om veribage
- Lextools rule-based system (also a perl version). Rules attempt to identify:
  - ⋆ Dates, times (various formats)
  - ⋆ telephone numbers
  - ⋆ fractions
  - ⋆ filenames/URL's,
  - ⋆ ordinals
  - ⋆ . . .

  Otherwise mixed alpha/non-alpha strings are split.

Finding Abbreviations and Potential Expansions: Dictionaries

- Large dictionary of ordinary words (320K words from U Penn XTag dictionary) augmented with 50K proper names.

  Outstanding problem: abbreviations can also be words – *kit* (*kitchen*); *abt* (*about*).

- Stoplist of things to leave alone. E.g.:

  nfcc, *rcam*, *att*, *cio*, *asap* . . .

  (Has same problem as above)

- If a token is (almost) purely alphabetic and it's not in the above list, treat it as a potential abbreviation

  Problem: some abbrevations use non-alphabetic symbols – *2 go*, *4x's*

Finding Abbreviations and Potential Expansions: Approximate Matching

- Collect bigrams of ordinary words.
- Collect token bigrams containing at least one potential abbreviation.
- Match abbreviation bigrams to word bigrams: e.g. *cus wnt* → *customer went*.

  Match potential abbreviation with full word if:
  - ⋆ Both start with same letter
  - ⋆ The abbreviation contains only letters and a few acceptable non-alphabetic symbols (', ., /)
  - ⋆ No letter in the abbreviation occurs more frequently than it does in the full form
  - ⋆ Letters in the abbreviation occur in (roughly) the same sequence as they do in the full form.

  So *ctsr* will match with *customer* but *clld* wouldn't.

Finding Abbreviations and Potential Expansions: Approximate Matching

- A few "phonetic" matches are allowed:

c	*see*
x-	*trans-, ex-*

- Some examples of matched pairs:

cus wnt	*customer went, customer wanted, customer wants*
bill pym	*bill payment*
insd wr	*inside wire, inside wiring, inside work*
pymnt argmnt	*payment arrangement, payment agreement, payment arrangements*
intrnt adlt	*internet adult*
line bld	*line blocked, line billed*

Language Modeling

- Train a word trigram model with standard Katz backoff on "sanitized" text:

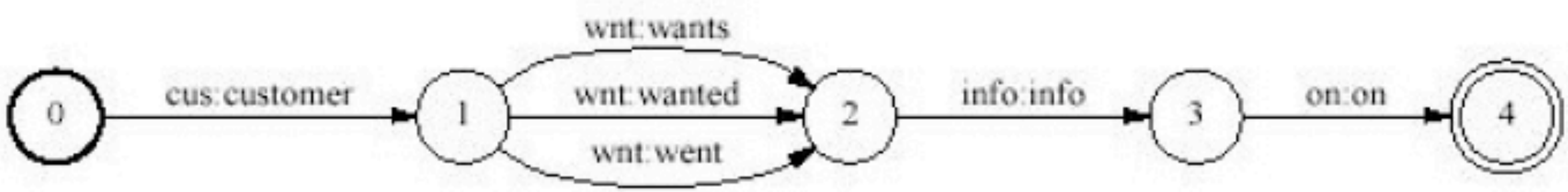
  cust business acct – trns to business office

  <ABBR> business <ABBR> <PUNC> <ABBR> to business office

- Implemented using the WFST-based LM tools that we've seen before

WFST-based Implementation

$$T' = \pi_2(ShortestPath(T \circ A^{-1} \circ L))$$



Further issues

- Run the normalization on the training data, treat the result as "truth", and reestimate the expansions of abbreviations; can also retrain the LM on the new "truth".

  This has been shown to reduce error rates by as much as 20% on classified ads.

  This allows one to reestimate each component term in:

  $p(\mathbf{o}|\mathbf{t},\mathbf{w})p(\mathbf{t}|\mathbf{w})p(\mathbf{w})$

- Does limiting the detection of abbreviations to bigrams that match full word bigrams help or hurt?

Some Example Normalizations (All RAMP Model)

cst cld 2 hv cllr id blck rmvn snt local form
customer called 2 have caller id block rmvn sent local form

cst clld to verify insde wre / i cncled his near mve on accident / cst now wnts to ploc to anther cmpny
customer called to verify inside wire / i cancelled his near move on accident / cst now wants to ploc to anther cmpny

cust no lnger wnts ld on acct
customer no longer wants ld on account

xplnd chrgs .. cust stated he w/ pay 26.45 & then w/ cancel his srvc w/ att
explained charges .. customer stated he will pay 26.45 & then will cancel his service with att

Back to Morphology

- Morphology is the study of the way words are built up from smaller meaning-bearing units, **morphemes**.
- Two broad classes of morphemes:
  - **The stems:** the "main" morpheme of the word, supplying the main meaning, while
  - **The affixes:** add "additional" meaning of various kinds.
- Affixes are further divided into **prefixes**, **suffixes**, **infixes**, and **circumfixes**.
  - Suffix: *eat-s*
  - Prefix: *un-buckle*
  - Circumfix: *ge-sag-t* (said) *sagen* (to say) (in German)
  - Infix: *hingi* (borrow) *h**um**ingi* (the agent of an action))in Philippine language Tagalog)

Survey of (Mostly) English Morphology

- Prefixes and suffixes are often called **concatenative morphology.**
- A number of languages have extensive **non-concatenative morphology**
  - The Tagalog infixation example
  - **Templatic morphology** or **root-and-pattern morphology**, common in Arabic, Hebrew, and other Semitic languages
- Two broad classes of ways to form words from morphemes:
  - **Inflection:** the combination of a word stem with a grammatical morpheme, usually resulting in a word of the same class as the original tem, and usually filling some syntactic function like agreement, and
  - **Derivation**: the combination of a word stem with a grammatical morpheme, usually resulting in a word of a *different* class, often with a meaning hard to predict exactly.

Survey of (Mostly) English Morphology

Inflectional Morphology

- In English, only nouns, verbs, and sometimes adjectives can be inflected, and the number of affixes is quite small.
- Inflections of nouns in English:
  - An affix marking **plural**,
    - `cat(-s), thrush(-es), ox (oxen), mouse (mice)`
    - `ibis(-es), waltz(-es), finch(-es), box(-es), butterfly(-lies)`
  - An affix marking **possessive**
    - `llama's, children's, llamas', Euripides' comedies`

Survey of (Mostly) English Morphology
Inflectional Morphology

- Verbal inflection is more complicated than nominal inflection.
  - English has three kinds of verbs:
    - **Main verbs**, *eat*, *sleep*, *impeach*
    - **Modal verbs**, *can will*, *should*
    - **Primary verbs**, *be*, *have*, *do*
  - Morphological forms of regular verbs

stem	walk	merge	try	map
-*s* form	walks	merges	tries	maps
-*ing* principle	walking	merging	trying	mapping
Past form or –*ed* participle	walked	merged	tried	mapped

– These regular verbs and forms are significant in the morphology of English because of their *majority* and being *productive*.

Survey of (Mostly) English Morphology
Inflectional Morphology

- Morphological forms of irregular verbs

stem	eat	catch	cut
-*s* form	eats	catches	cuts
-*ing* principle	eating	catching	cutting
Past form	ate	caught	cut
–*ed* participle	eaten	caught	cut

Survey of (Mostly) English Morphology
Derivational Morphology

- **Nominalization** in English:
  - The formation of new nouns, often from verbs or adjectives

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

– Adjectives derived from nouns or verbs

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

Survey of (Mostly) English Morphology

Derivational Morphology

- Derivation in English is more complex than inflection because
  - Generally less productive
    - A nominalizing affix like –*ation* can not be added to absolutely every verb. *eatation*(*)
  - There are subtle and complex meaning differences among nominalizing suffixes. For example, *sincerity* has a subtle difference in meaning from *sincereness*.

Finite-State Morphological Parsing

- Parsing English morphology

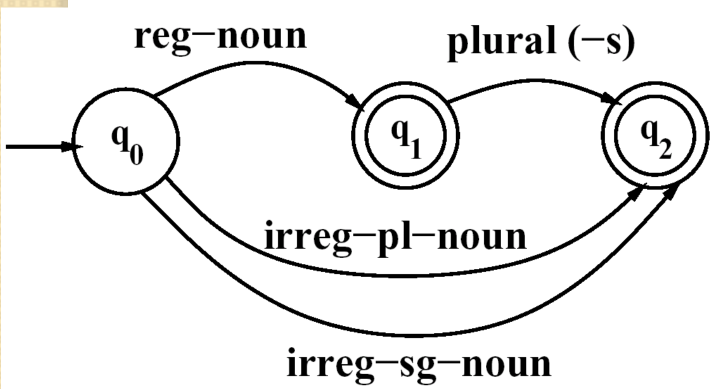
Input	Morphological parsed output
cats	`cat +N +PL`
cat	`cat +N +SG`
cities	`city +N +PL`
geese	`goose +N +PL`
goose	(`goose +N +SG`) or (`goose +V`)
gooses	`goose +V +3SG`
merging	`merge +V +PRES-PART`
caught	(`caught +V +PAST-PART`) or (`catch +V +PAST`)

Finite-State Morphological Parsing

- We need at least the following to build a morphological parser:
  1. **Lexicon**: the list of stems and affixes, together with basic information about them (Noun stem or 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. E.g., the rule that English plural morpheme follows the noun rather than preceding it.
  3. **Orthographic rules**: these **spelling rules** are used to model the changes that occur in a word, usually when two morphemes combine (e.g., the *y*→*ie* spelling rule changes *city* + *-s* to *cities*).

Finite-State Morphological Parsing
The Lexicon and Morphotactics

- A lexicon is a repository for words.
  - The simplest one would consist of an explicit list of every word of the language. ***Incovenient or impossible!***
  - Computational lexicons are usually structured with
    - a list of each of the stems and
    - Affixes of the language together with a representation of morphotactics telling us how they can fit together.
  - The most common way of modeling morphotactics is the finite-state automaton.

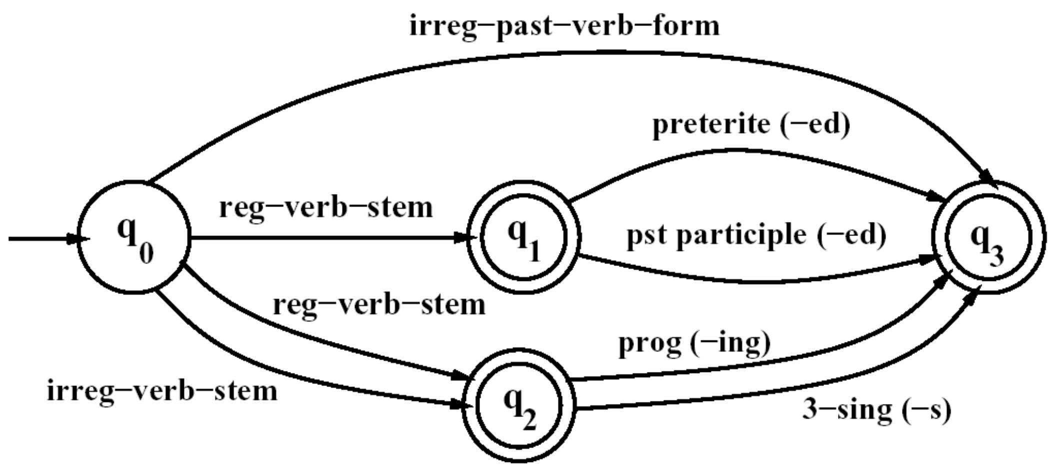


An FSA for English nominal inflection

Reg-noun	Irreg-pl-noun	Irreg-sg-noun	plural
fox fat fog fardvark	geese sheep Mice	goose sheep mouse	-s

Finite-State Morphological Parsing

The Lexicon and Morphotactics



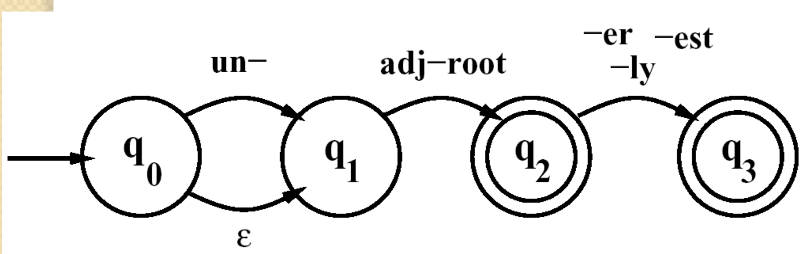
An FSA for English verbal inflection

Reg-verb-stem	Irreg-verb-stem	Irreg-past-verb	past	Past-part	Pres-part	3sg
walk fry talk impeach	cut speak sing sang spoken	caught ate eaten	-ed	-ed	-ing	-s

Finite-State Morphological Parsing

The Lexicon and Morphotactics

- English derivational morphology is more complex than English inflectional morphology, and so automata of modeling English derivation tends to be quite complex.
  - Some even based on CFG
- A small part of morphosyntactics of English adjectives

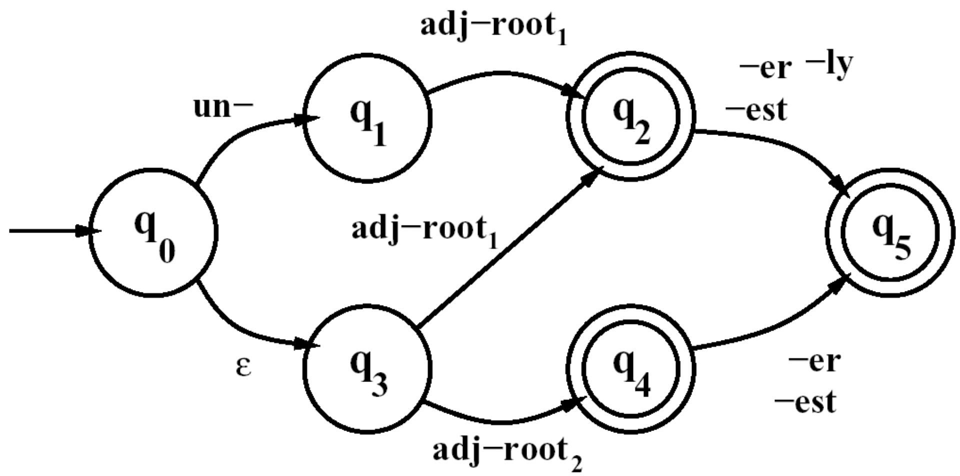


An FSA for a fragment of English adjective Morphology #1

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

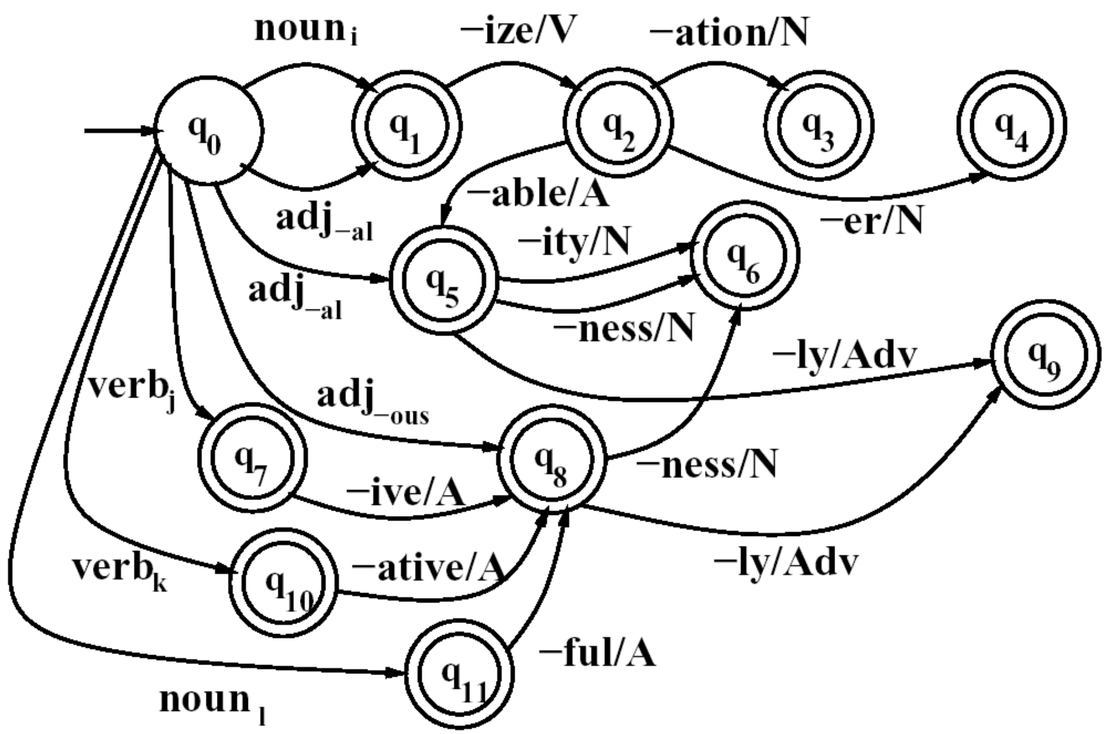
Finite-State Morphological Parsing

- The FSA#1 recognizes all the listed adjectives, and ungrammatical forms like *unbig, redly,* and *realest.*
- Thus #1 is revised to become #2.
- The complexity is expected from English derivation.



An FSA for a fragment of English adjective Morphology #2

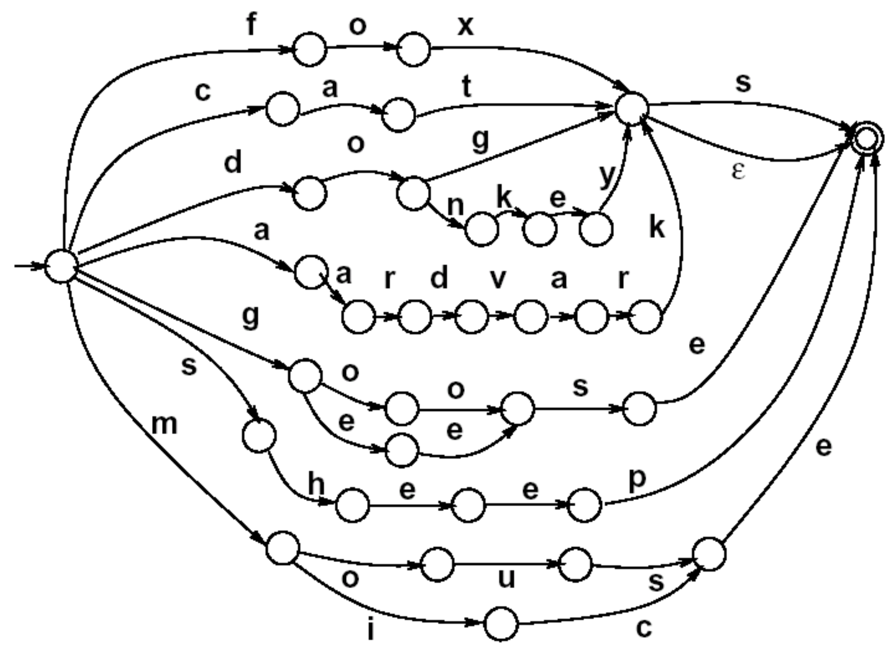
Finite-State Morphological Parsing



An FSA for another fragment of English derivational morphology

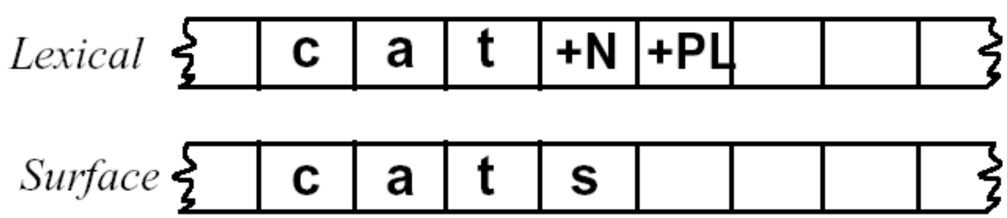
Finite-State Morphological Parsing

- We can now use these FSAs to solve the problem of **morphological recognition**:
  - Determining whether an input string of letters makes up a legitimate English word or not
  - We do this by taking the morphotactic FSAs, and plugging in each "sub-lexicon" into the FSA.
  - The resulting FSA can then be defined as the level of the individual letter.



Finite-State Morphological Parsing
Morphological Parsing with FST

- Given the input, for example, *cats*, we would like to produce `cat +N +PL`.
- Two-level morphology, by Koskenniemi (1983)
  - Representing a word as a correspondence between a **lexical level**
    - Representing a simple concatenation of morphemes making up a word, and
  - The **surface level**
    - Representing the actual spelling of the final word.
- Morphological parsing is implemented by building mapping rules that maps letter sequences like *cats* on the surface level into morpheme and features sequence like `cat +N +PL` on the lexical level.



Finite-State Morphological Parsing

Morphological Parsing with FST

- The automaton we use for performing the mapping between these two levels is the **finite-state transducer** or **FST**.
  - A transducer maps between one set of symbols and another;
  - An FST does this via a finite automaton.
- Thus an FST can be seen as a two-tape automaton which **recognizes** or **generates** ***pairs*** of strings.
- The FST has a more general function than an FSA:
  - An FSA defines a formal language
  - An FST defines a relation between sets of strings.
- Another view of an FST:
  - A machine reads one string and generates another.

Finite-State Morphological Parsing

Morphological Parsing with FST

- **FST as recognizer:**
  - a transducer that takes a pair of strings as input and output *accept* if the string-pair is in the string-pair language, and a *reject* if it is not.
- **FST as generator:**
  - a machine that outputs pairs of strings of the language. Thus the output is a yes or no, and a pair of output strings.
- **FST as transducer:**
  - A machine that reads a string and outputs another string.
- **FST as set relater:**
  - A machine that computes relation between sets.

Finite-State Morphological Parsing

Morphological Parsing with FST

- A formal definition of FST (based on the **Mealy machine** extension to a simple FSA):
  - Q: a finite set of N states $q_0, q_1, \ldots, q_N$
  - Σ: a finite alphabet of complex symbols. Each complex symbol is composed of an input-output pair $i : o$; one symbol I from an input alphabet I, and one symbol o from an output alphabet O, thus $\Sigma \subseteq I \times O$. I and O may each also include the epsilon symbol ε.
  - q_0: the start state
  - F: the set of final states, $F \subseteq Q$
  - $\delta(q, i{:}o)$: the transition function or transition matrix between states. Given a state $q \in Q$ and complex symbol $i{:}o \in \Sigma$, $\delta(q, i{:}o)$ returns a new state $q' \in Q$. δ is thus a relation from $Q \times \Sigma$ to Q.

Finite-State Morphological Parsing

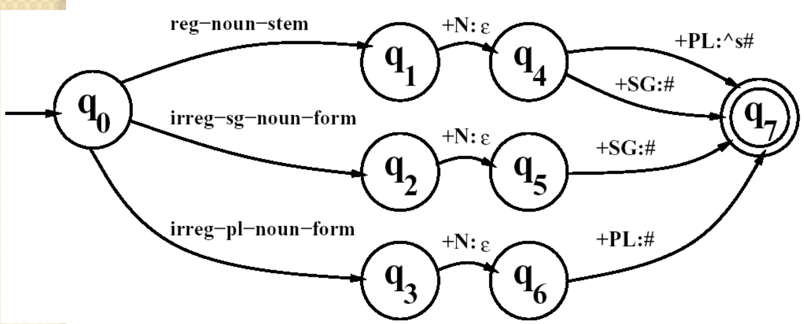
Morphological Parsing with FST

- FSAs are isomorphic to regular languages, FSTs are isomorphic to **regular relations.**
- Regular relations are sets of pairs of strings, a natural extension of the regular language, which are sets of strings.
- FSTs are closed under union, but generally they are not closed under difference, complementation, and intersection.
- Two useful closure properties of FSTs:
  - **Inversion:** If T maps from I to O, then the inverse of T, T^{-1} maps from O to I.
  - **Composition:** If T_1 is a transducer from I_1 to O_1 and T_2 a transducer from I_2 to O_2, then $T_1 \circ T_2$ maps from I_1 to O_2

Finite-State Morphological Parsing

Morphological Parsing with FST

- Inversion is useful because it makes it easy to convert a FST-as-parser into an FST-as-generator.
- Composition is useful because it allows us to take two transducers than run in series and replace them with one complex transducer.
  - $T_1 \circ T_2(S) = T_2(T_1(S))$

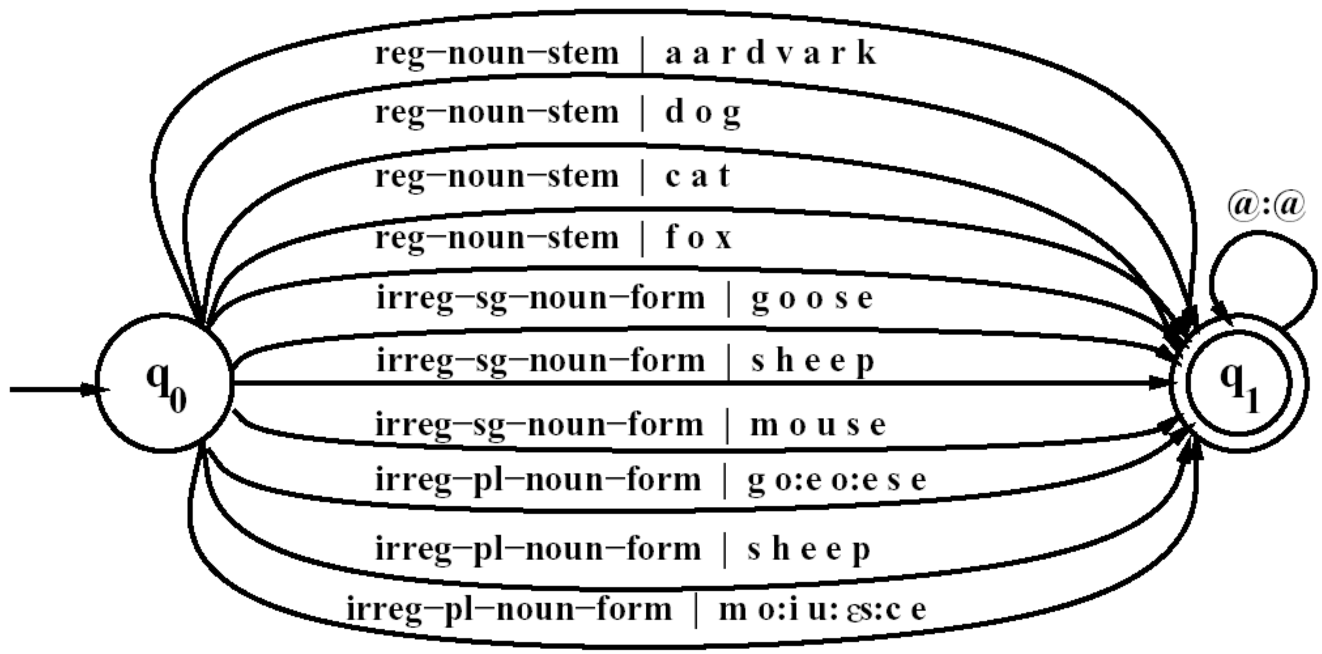


A transducer for English nominal number inflection T_{num}

Reg-noun	Irreg-pl-noun	Irreg-sg-noun
fox	g o:e o:e s e	goose
fat	sheep	sheep
fog	m o:i u:εs:c e	mouse
aardvark		

Finite-State Morphological Parsing

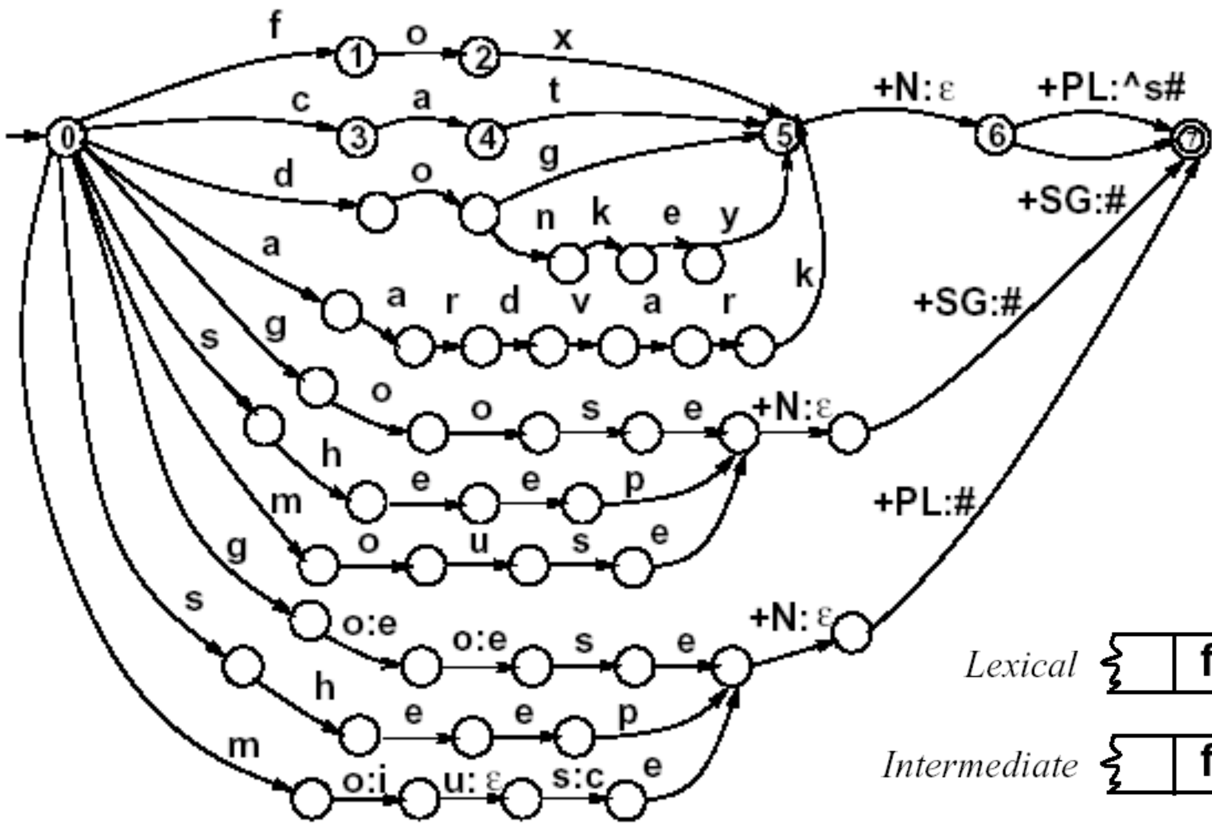
Morphological Parsing with FST



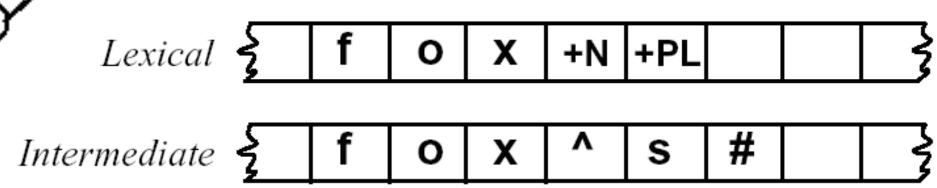
The transducer T_{stems}, which maps roots to their root-class

Finite-State Morphological Parsing

Morphological Parsing with FST



^: morpheme boundary
#: word boundary



A fleshed-out English nominal inflection FST

$T_{lex} = T_{num} \circ\ T_{stems}$

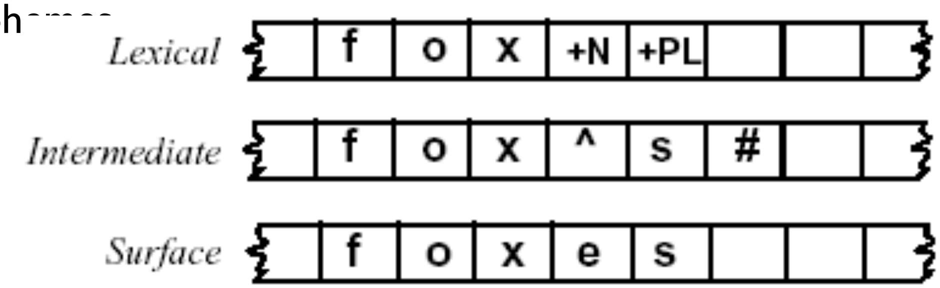
Finite-State Morphological Parsing

Orthographic Rules and FSTs

- **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.



Finite-State Morphological Parsing

Orthographic Rules and FSTs

- "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*"

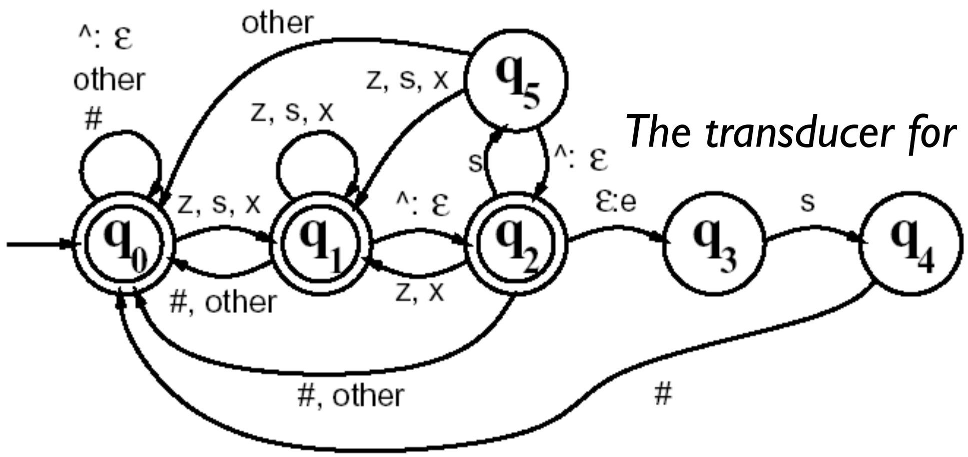
$$\varepsilon \rightarrow e / \left\{ \begin{array}{c} x \\ s \\ z \end{array} \right\} \_\_ \; s\#$$

- "rewrite *a* to *b* when it occurs between *c* and *d*"

$$a \rightarrow b \; / \; c \_\_ \; d$$

Finite-State Morphological Parsing

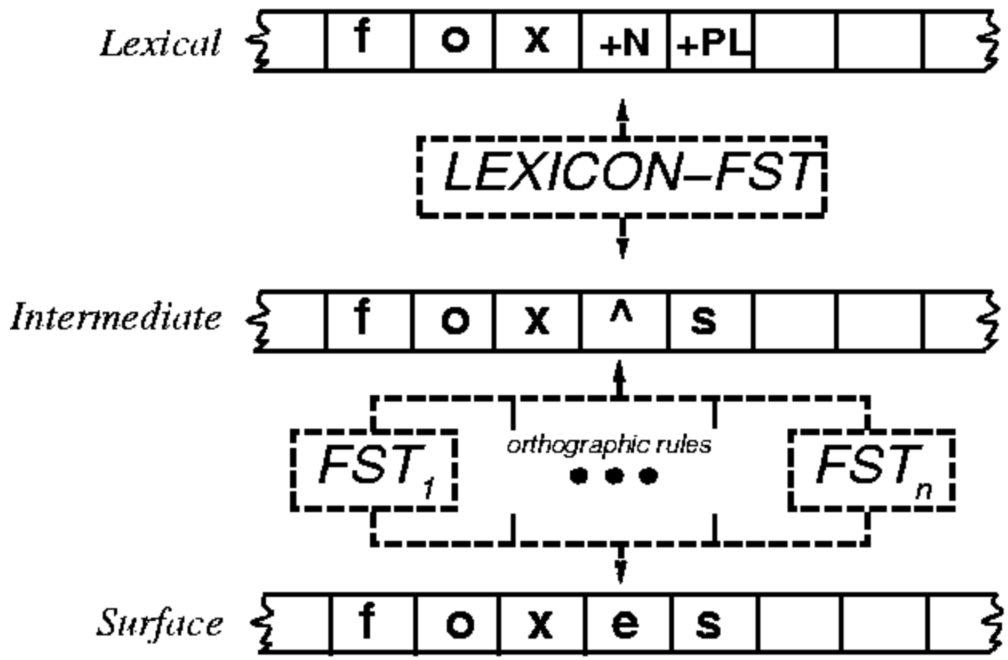
Orthographic Rules and FSTs



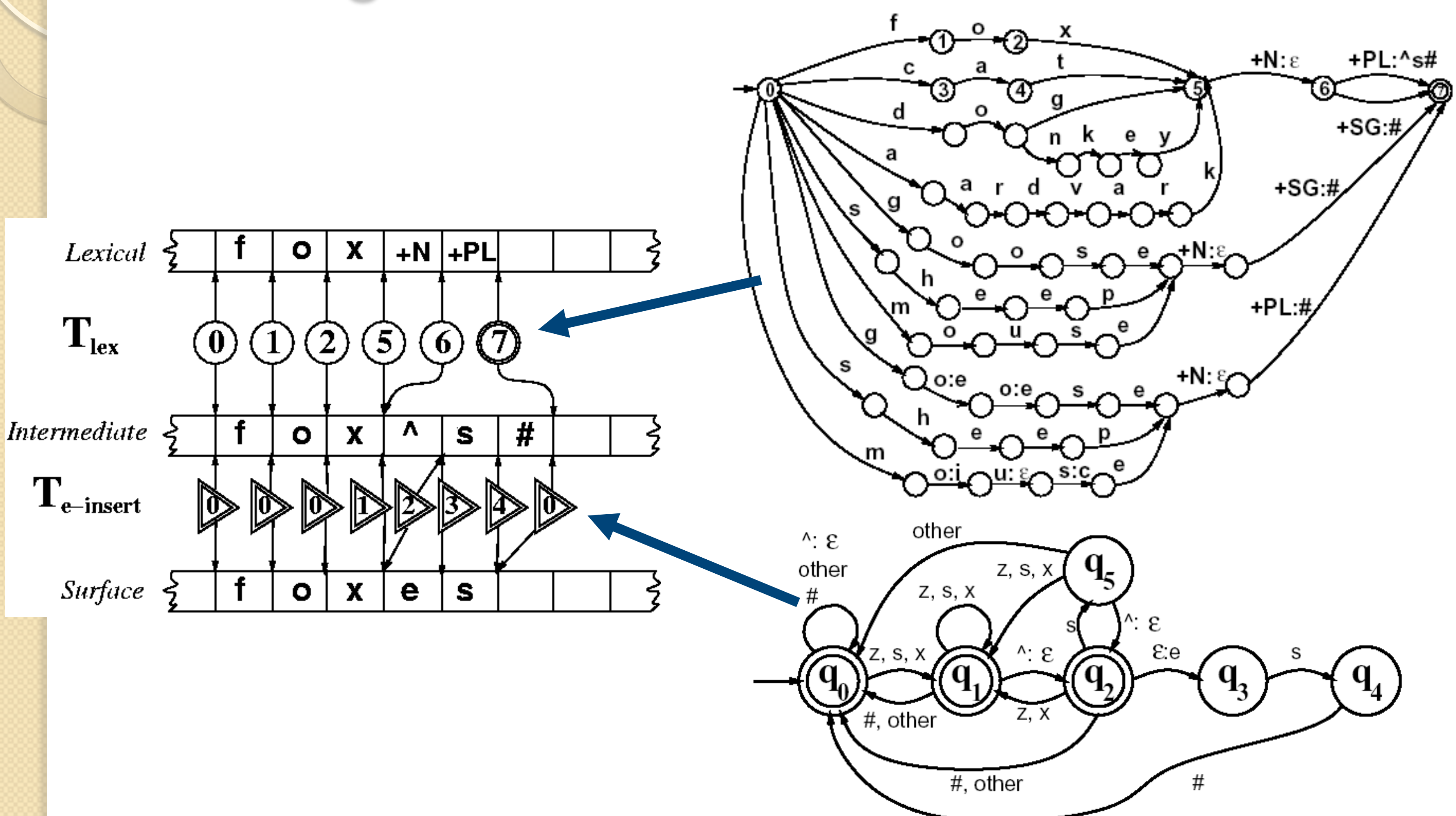
The transducer for the E-insertion rule

State \ Input	s:s	x:x	z:z	^:ε	ε:e	#	other
q_0:	1	1	1	0	-	0	0
q_1:	1	1	1	2	-	0	0
q_2:	5	1	1	0	3	0	0
q_3	4	-	-	-	-	-	-
q_4	-	-	-	-	-	0	-
q_5	1	1	1	2	-	-	0

Combining FST Lexicon and Rules



Combining FST Lexicon and Rules



Combining FST Lexicon and Rules

- The power of FSTs is that the exact same cascade with the same state sequences is used
  - when machine is generating the surface form from the lexical tape, or
  - When it is parsing the lexical tape from the surface tape.
- Parsing can be slightly more complicated than generation, because of the problem of **ambiguity.**
  - For example, *foxes* could be `fox +V +3SG` as well as `fox +N +PL`

Lexicon-Free FSTs: the Porter Stemmer

- Information retrieval
- One of the mostly widely used stemming algorithms is the simple and efficient Porter (1980) algorithm, which is based on a series of simple cascaded rewrite rules.
  - ATIONAL → ATE (e.g., relational → relate)
  - ING → ε if stem contains vowel (e.g., motoring → motor)
- Problem:
  - Not perfect: error of commision, omission
- Experiments have been made
  - Some improvement with smaller documents
  - Any improvement is quite small