Accelerated Natural Language Processing
Lecture 3
Morphology and Finite State Machines; Edit Distance

Sharon Goldwater
(based on slides by Philipp Koehn)

20 September 2019

School of informatics

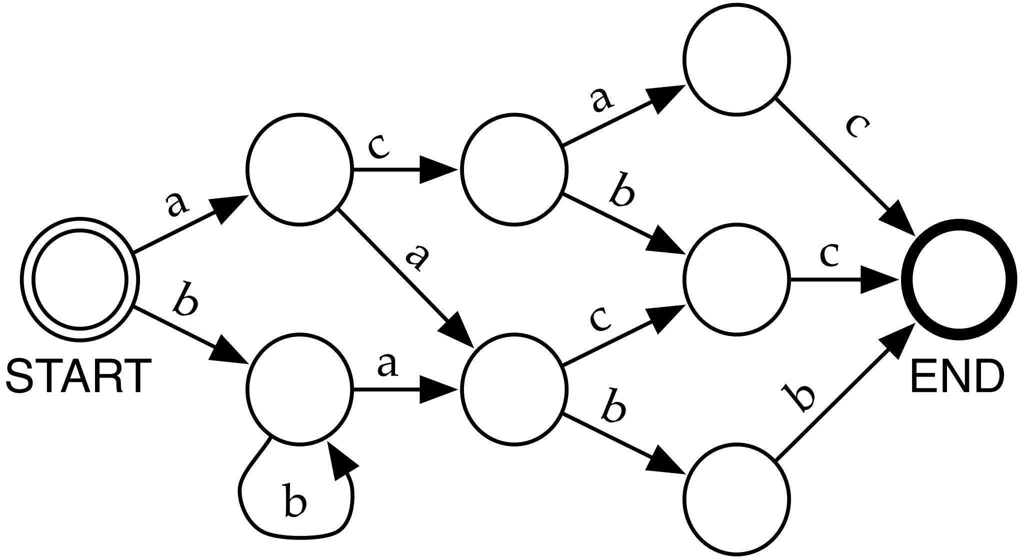
Recap: Tasks

- Recognition
  - given: surface form
  - wanted: yes/no decision if it is in the language
- Generation
  - given: lemma and morphological properties
  - wanted: surface form
- Analysis
  - given: surface form
  - wanted: lemma and morphological properties

Recap: General approach

- Could list all words with their analyses, but
  - List gets too big
  - Language is infinite, cannot generalize beyond list
- Instead, use finite state machines
  - Finite and compact representation of infinite language
  - Several toolkits available

Recap: Finite State Automata



Can be viewed as either *emitting* or *recognizing* strings

Today's lecture

- How are FSMs and FSTs used for morphological recognition, analysis and generation?

- How can we deal with spelling changes in morphological analysis?

- What is an alignment between two strings?

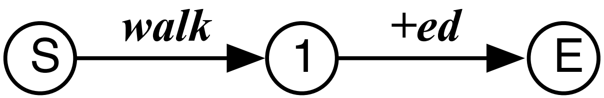
- What is minimum edit distance and how do we compute it? What's wrong with a brute force solution, and how do we solve that problem?

One Word

Basic finite state automaton:

- start state
- transition that emits the word walk
- end state

One Word and One Inflection

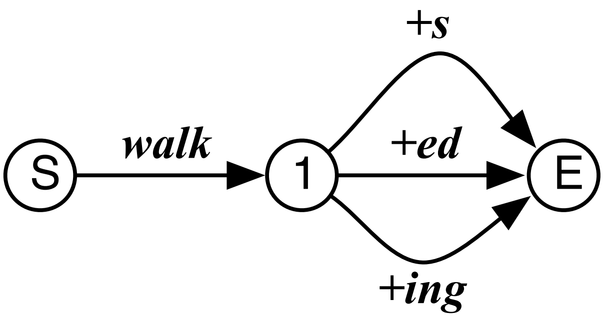


Two transitions and intermediate state

- first transition emits walk
- second transition emits +ed

$\rightarrow$ walked

One Word and Multiple Inflections

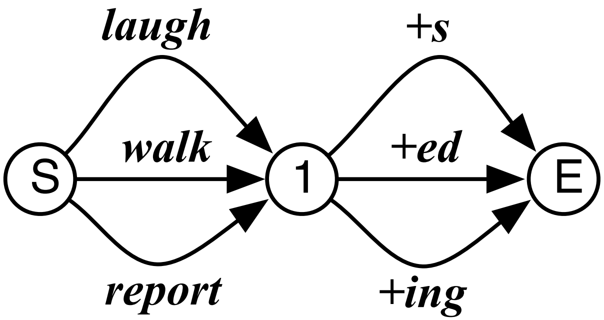


Multiple transitions between states

- three different paths

$\rightarrow$ walks, walked, walking

Multiple Words and Multiple Inflections

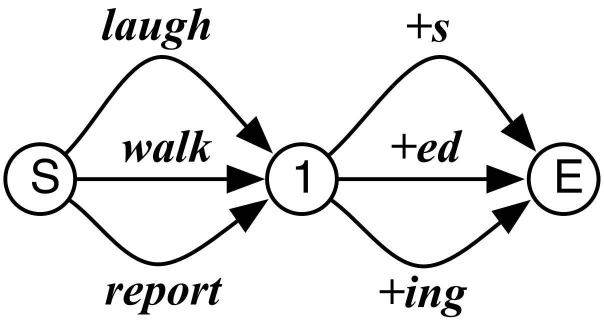


Multiple stems

- implements regular verb morphology

$\rightarrow$ laughs, laughed, laughing
walks, walked, walking
reports, reported, reporting

Multiple Words and Multiple Inflections



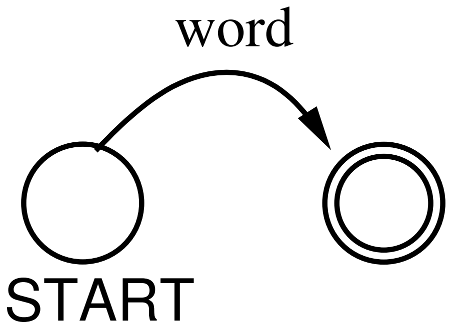
Multiple stems

- implements regular verb morphology

$\rightarrow$ what about bake, emit, fuss?

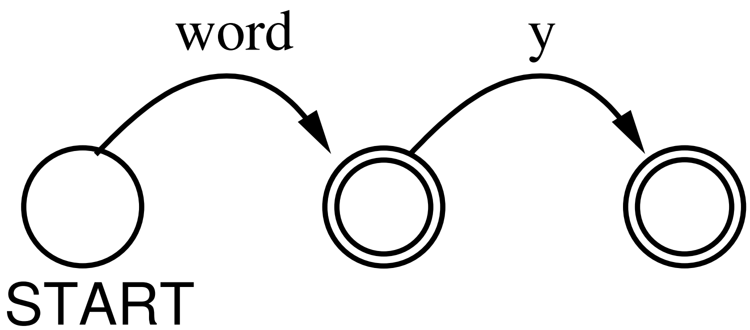
more on this later...

Derivational Morphology

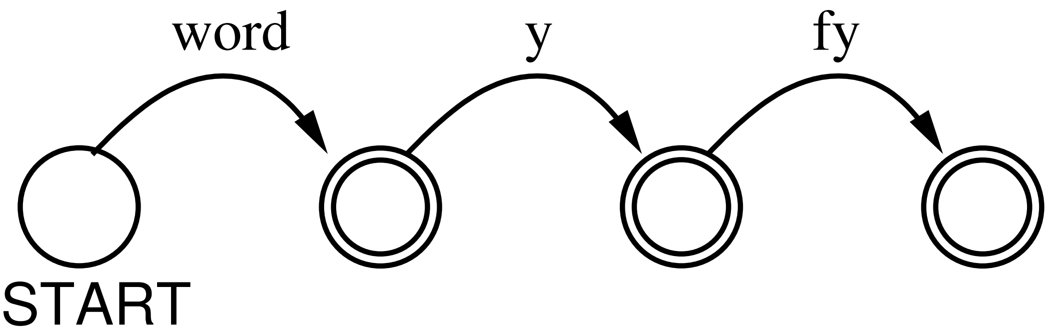


double lines = end state

Derivational Morphology



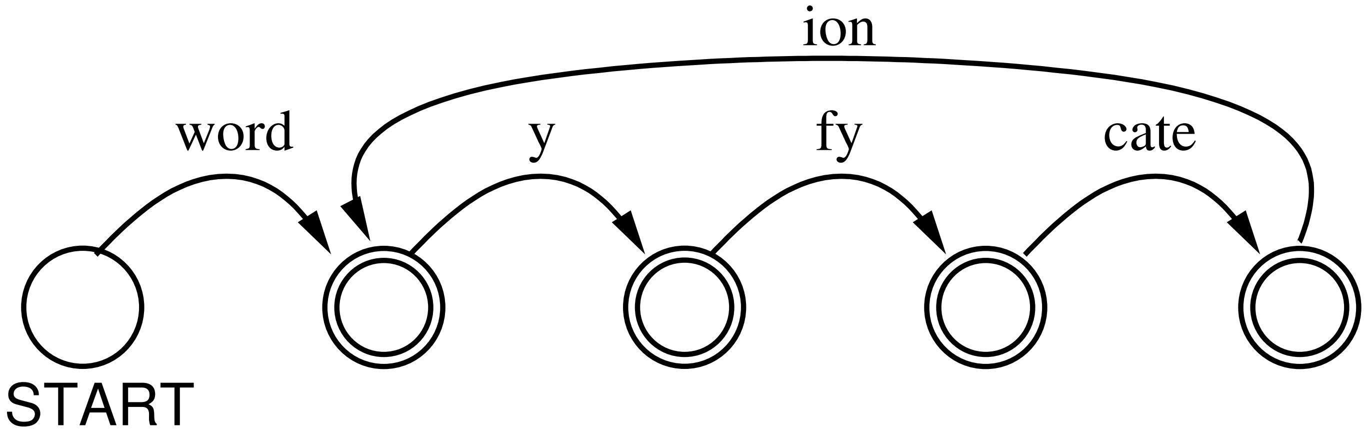
Derivational Morphology



again: wordify not wordyfy!

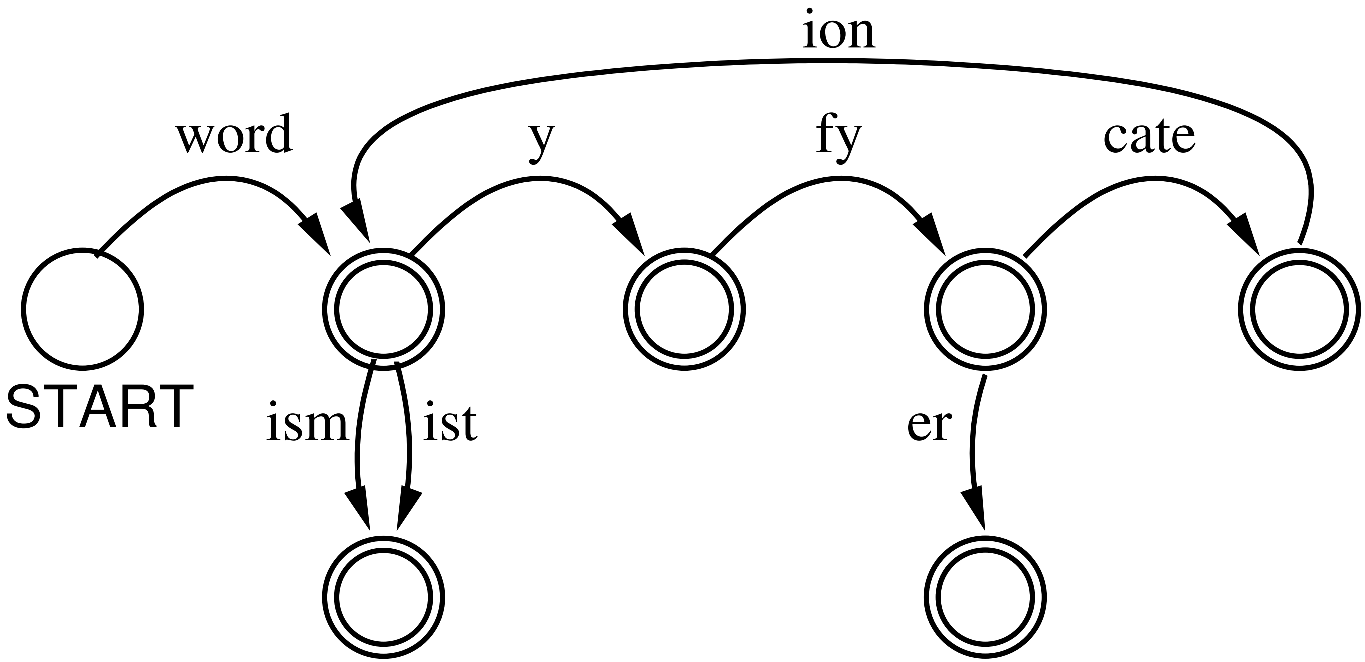
again, will come back to that later...

Derivational Morphology

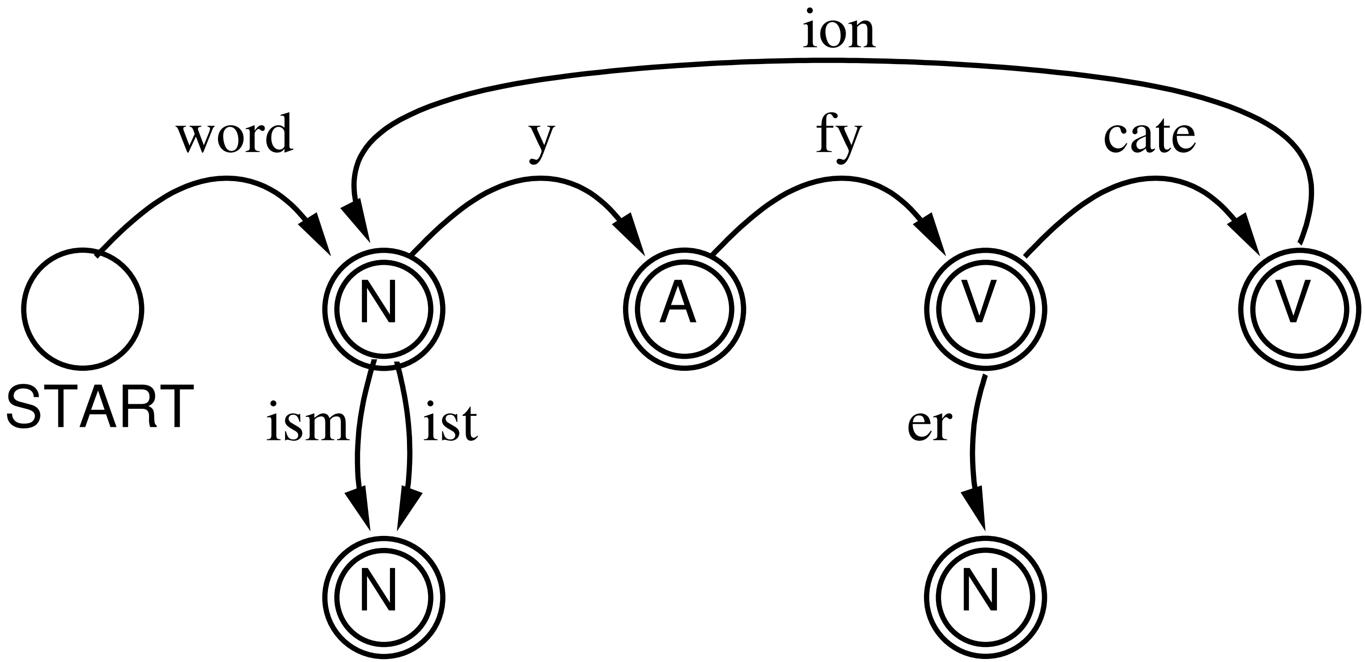


why a loop? could it be placed differently?

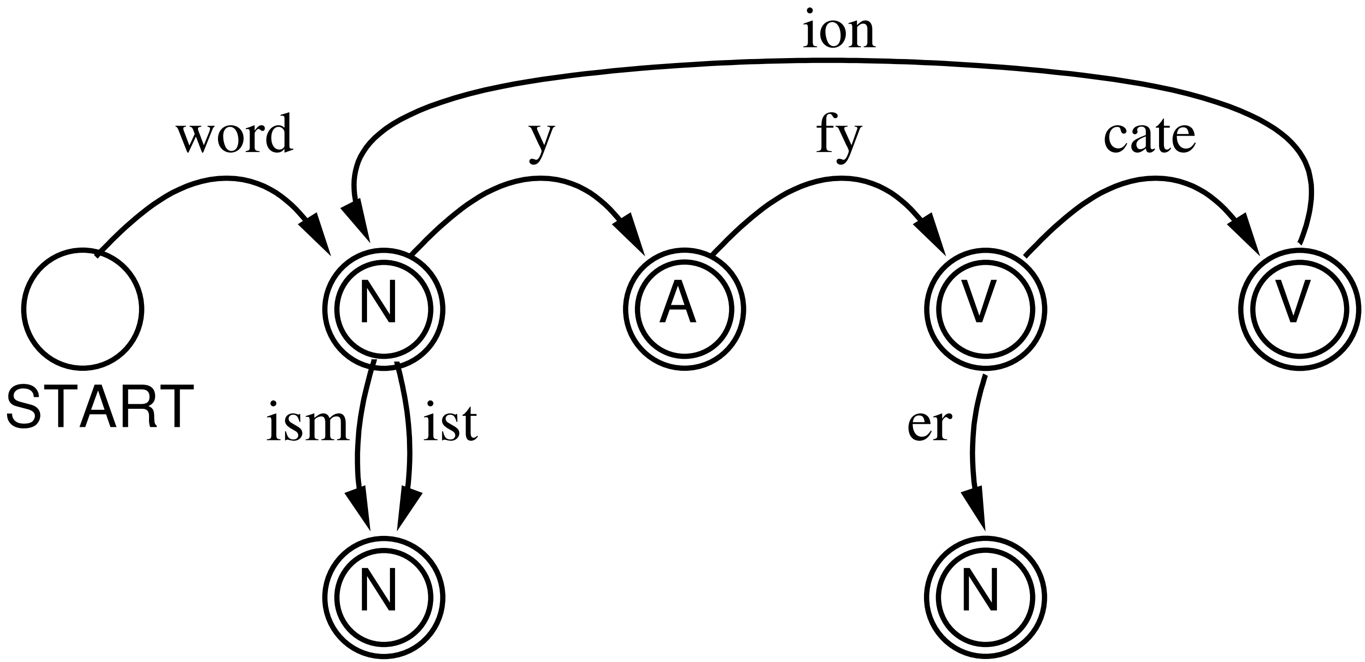
Derivational Morphology



Marking Part of Speech



Marking Part of Speech



Now: where to add -less? -ness? Others?

Concatenation

- Constructing an FSA gets very complicated

- Build components as separate FSAs
  - **L**: FSA for lexicon
  - **D**: FSA for derivational morphology
  - **I**: FSA for inflectional morphology

- Concatenate **L** + **D** + **I** (there are standard algorithms)
  - In fact, each component may consist of multiple components (e.g., **D** has different sets of affixes with ordering constraints)

What is Required?

- Lexicon of lemmas
  - very large, needs to be collected by hand
- Inflection and derivation rules
  - not large, but requires understanding of the language

Recent work: automatically learn lemmas and suffixes from a corpus

- OK solution for languages with few resources
- Hand-engineered systems much better when available

Generation and analysis

- FSAs used as morphological *recognizers*

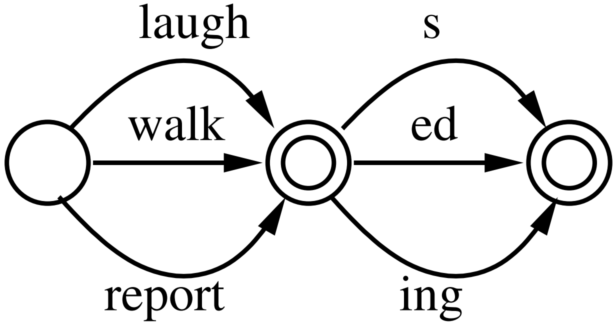
- What if we want to generate or analyze?

$$\text{walk+V+past} \leftrightarrow \text{walked}$$
$$\text{report+V+prog} \leftrightarrow \text{reporting}$$

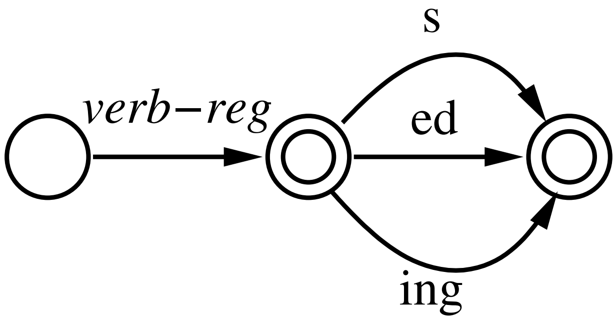
- Use a finite-state *transducer* (FST)

  – Replace *output* symbols with *input-output* pairs $x : y$

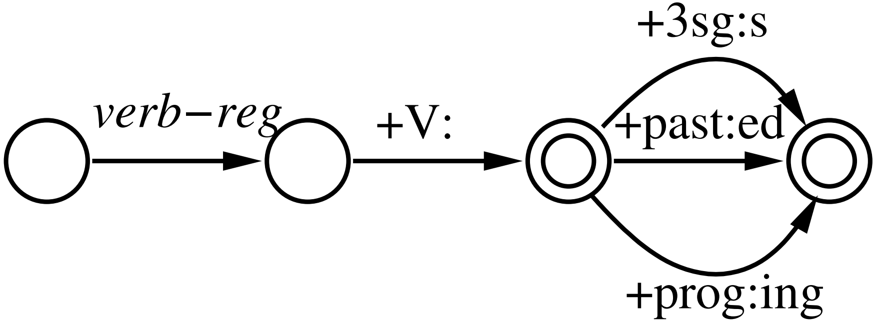
FSA for verbs



Schematically



FST for verbs



where x means x:x and x: means x:ϵ.

Accounting for spelling changes

- We now have:

$$\text{walk+V+past} \leftrightarrow \text{walked}$$

BUT

$$\text{bake+V+past} \leftrightarrow \text{bakeed}$$

- How to fix this?

Accounting for spelling changes

- We now have:

$$\text{walk+V+past} \leftrightarrow \text{walked}$$

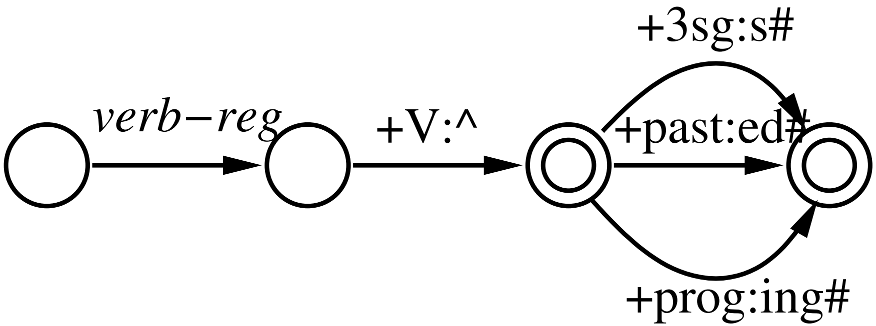
BUT

$$\text{bake+V+past} \leftrightarrow \text{bakeed}$$

- How to fix this? Use *two* FSTs in a row!

$$\text{walk+V+past} \leftrightarrow \text{walk\^{}ed\#} \leftrightarrow \text{walked}$$
$$\text{bake+V+past} \leftrightarrow \text{bake\^{}ed\#} \leftrightarrow \text{baked}$$

1. Analysis to intermediate form

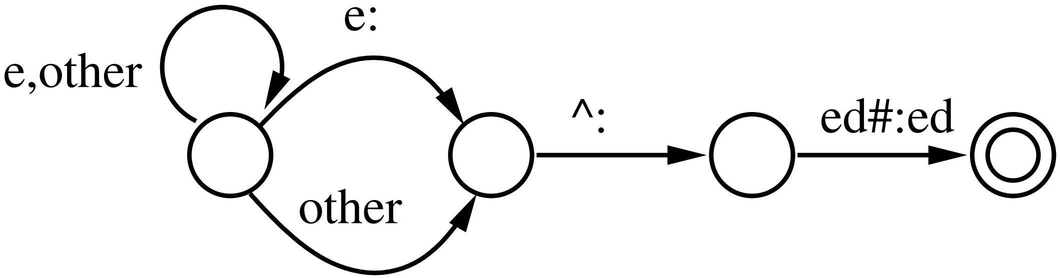


where x means x:x and x: means x:ϵ.

- Examples:
  walk+V+past ↔ walk^ed#
  bake+V+past ↔ bake^ed#
  bake+V+prog ↔ bake^ing#

2. Intermediate form to surface form

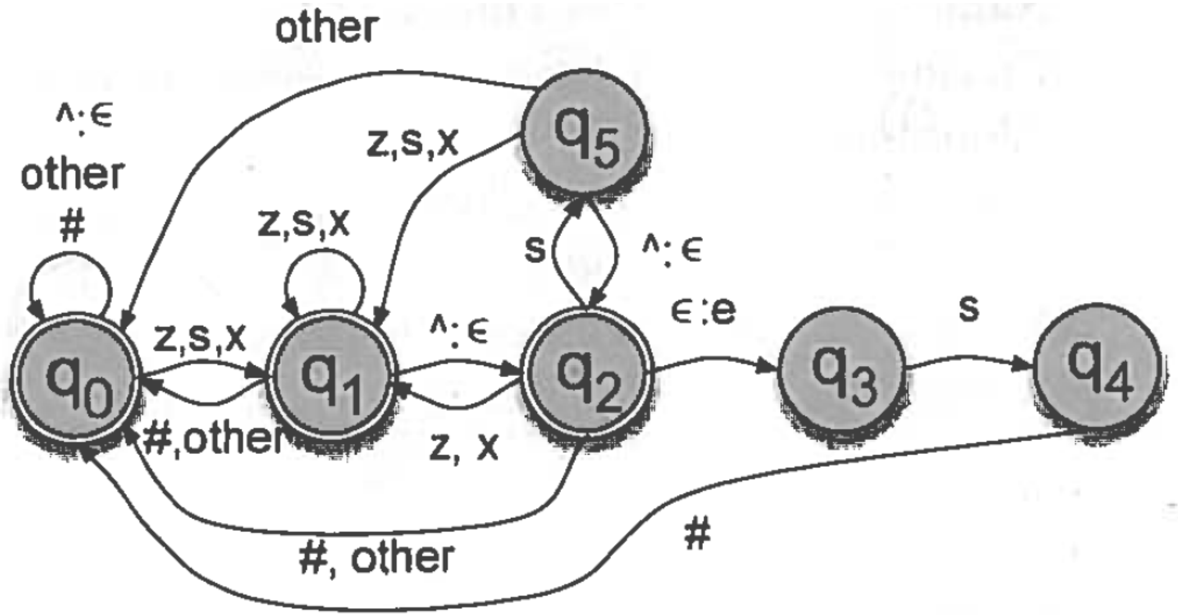
Simplified version, only handles some aspects of past tense:



where $other$ means any character except 'e'.

- Examples: walkˆed# ↔ walked, bakeˆed# ↔ baked
- A **nondeterministic** FST: mulitple transitions may be possible on the same input (where?).
  If *any* path goes to end state, string is accepted.

Plural transducer (J&M, Fig. 3.17)



- Complete FST for English plural ('other' = none of {z,s,x,^,#,ϵ})
- What happens in each case? cat^s# fox^s# axle^s#

Remaining problem: ambiguity

- FSTs often produce multiple analyses for a single form:

  walks $\rightarrow$ walk+V+1sg OR walk+N+pl

  German 'the': 6 surface forms, but 24 possible analyses

- Resolve using *context* (surrounding words), usually in a probabilistic system (stay tuned...)

More info and tools

- More information: Oflazer (2009): Computational Morphology

  `http://fsmnlp2009.fastar.org/Program_files/Oflazer - slides.pdf`

- OpenFST (Google and NYU)

  `http://www.openfst.org/`

- Carmel Toolkit

  `http://www.isi.edu/licensed-sw/carmel/`

- FSA Toolkit

  `http://www-i6.informatik.rwth-aachen.de/∼kanthak/fsa.html`

Related task: string similarity

Given two strings, how "similar" are they?

- Could indicate morphological relationships:

walk - walks, sleep - slept

- Or possible spelling errors (and corrections):

definition - defintion, separate - seperate

- Also used in other fields, e.g., bioinformatics:

ACCGTA - ACCGATA

One measure: minimum edit distance

- How many changes to go from string $s_1 \to s_2$?

S	T	A	L	L	
	T	A	L	L	deletion
	T	A	B	L	substitution
	T	A	B	L E	insertion

- To solve the problem, we need to find the best **alignment** between the words.
  - Could be several equally good alignments.

Example alignments

Let ins/del cost (distance) = 1, sub cost = 2 (0 if no change)
(can use other costs, incl diff costs for diff chars)

- Two optimal alignments (cost = 4):

S	T	A	L	L	-		S	T	A	-	L	L
d	|	|	s	|	i		d	|	|	i	|	s
-	T	A	B	L	E		-	T	A	B	L	E

Example alignments

Let ins/del cost (distance) = 1, sub cost = 2 (0 if no change)
(can use other costs, incl diff costs for diff chars)

- Two optimal alignments (cost = 4):

```
S T A L L -        S T A - L L
d | | s | i        d | | i | s
- T A B L E        - T A B L E
```

- LOTS of non-optimal alignments, such as:

```
S T A - L - L      S T A L - L -
s d | i | i d      d d s s i | i
T - A B L E -      - - T A B L E
```

Brute force solution: too slow

How many possible alignments to consider?

- First character could align to any of:

  `-  -  -  -  -  T  A  B  L  E  -`

- Next character can align anywhere to its right
- And so on... the number of alignments grows exponentially with the length of the sequences.

Brute force solution: too slow

How many possible alignments to consider?

- First character could align to any of:

- - - - - T A B L E -

- Next character can align anywhere to its right
- And so on... the number of alignments grows exponentially with the length of the sequences.

To solve, we use a *dynamic programming* algorithm

- Store solutions to smaller computations and combine them
- Widespread in NLP, e.g. tagging (HMMs), parsing (CKY)

Intuition

- Minimum distance D(stall, table) must be the minimum of:
  - D(stall, tabl) + 1 (ins e)
  - D(stal, table) + 1 (del l)
  - D(stal, tabl) + 2 (sub l/e)

- Similarly for the smaller subproblems

- So proceed as follows:
  - solve smallest subproblems first
  - store solutions in a table (chart)
  - use these to solve and store larger subproblems until we get the full solution

Chart: starting point

		T	A	B	L	E
	0	1	2	3	4	5
S	1					
T	2					
A	**3**					
L	4					
L	5					

- Chart[i, j] stores D(stall[0..i],table[0..j])
  - Ex: **Chart[3,0]** $=$ D(stall[0..3], table[0..0]) $=$ D(sta, ϵ)

Chart: next step

		T	A	B	L	E
	0	1	2	3	4	5
S	1	?				
T	2					
A	3					
L	4					
L	5					

- **Chart[1, 1]** is $\mathrm{D}(\mathrm{S},\ \mathrm{T})$: the minimum of

  $\mathrm{D}(\text{-},\ \mathrm{T}) + 1$ (**Chart[0, 1]** $+ 1 = 2$)

  $\mathrm{D}(\mathrm{S},\ \text{-}) + 1$ (**Chart[1, 0]** $+ 1 = 2$)

  $\mathrm{D}(\text{-},\ \text{-}) + 2$ (**Chart[0, 0]** $+ 2 = 2$)

Chart: one more step

		T	A	B	L	E
	0	1	2	3	4	5
S	1	2				
T	2	?				
A	3					
L	4					
L	5					

- **Chart[2, 1]** is $\mathrm{D(ST, T)}$: the minimum of

  $\mathrm{D(S, T)} + 1$ (**Chart[1, 1]** $+ 1 = 3$)

  $\mathrm{D(ST, -)} + 1$ (**Chart[2, 0]** $+ 1 = 3$)

  $\mathrm{D(S, -)} + 0$ (**Chart[1, 0]** $+ 0 = 1$)

Chart: next steps

		T	A	B	L	E
	0	1	2	3	4	5
S	1	2				
T	2	1				
A	3					
L	4					
L	5					

- Continue by filling in each full column in order (or go by rows)

Chart: completed

		T	A	B	L	E
	0	1	2	3	4	5
S	1	2	3	4	5	6
T	2	1	2	3	4	5
A	3	2	1	2	3	4
L	4	3	2	3	2	3
L	5	4	3	4	3	4

To find alignments

		T	A	B	L	E
	0	←1	←2	←3	←4	←5
S	↑1	←↖↑2	←↖↑3	←↖↑4	←↖↑5	←↖↑6
T	↑2	↖1	←2	←3	←4	←5
A	↑3	↑2	↖1	←2	←3	←4
L	↑4	↑3	↑2	←↖↑3	↖2	←3
L	↑5	↑4	↑3	←↖↑4	↖↑3	←↖↑4

- also store *which subproblem* the best score came from
- backtrack to get the best alignment

⇒ More complete worked example on lecture page, with exercises.

Questions for review

- How are FSMs and FSTs used for morphological recognition, analysis and generation?
- How can we deal with spelling changes in morphological analysis?
- What is an alignment between two strings?
- What is minimum edit distance and how do we compute it? What's wrong with a brute force solution, and how do we solve that problem?

Announcements

- Next lecture: Probability models and estimation.
  - Assumes you know or are getting to grips with the material in the maths tutorials on the Readings section. Do it now!
- Tutorials start on Tuesday. Try to register for this class ASAP so you're assigned to a group. But if not, go to a group anyway!
  - Work through the exercise sheet for Week 2 (see web page): bring questions to your tutorial group.
- Remember to register for Piazza (use link on Learn).
- Drop-in TA hours Mon/Fri 10-11, see Help+Support on Learn.