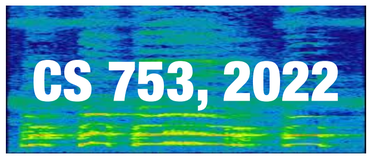
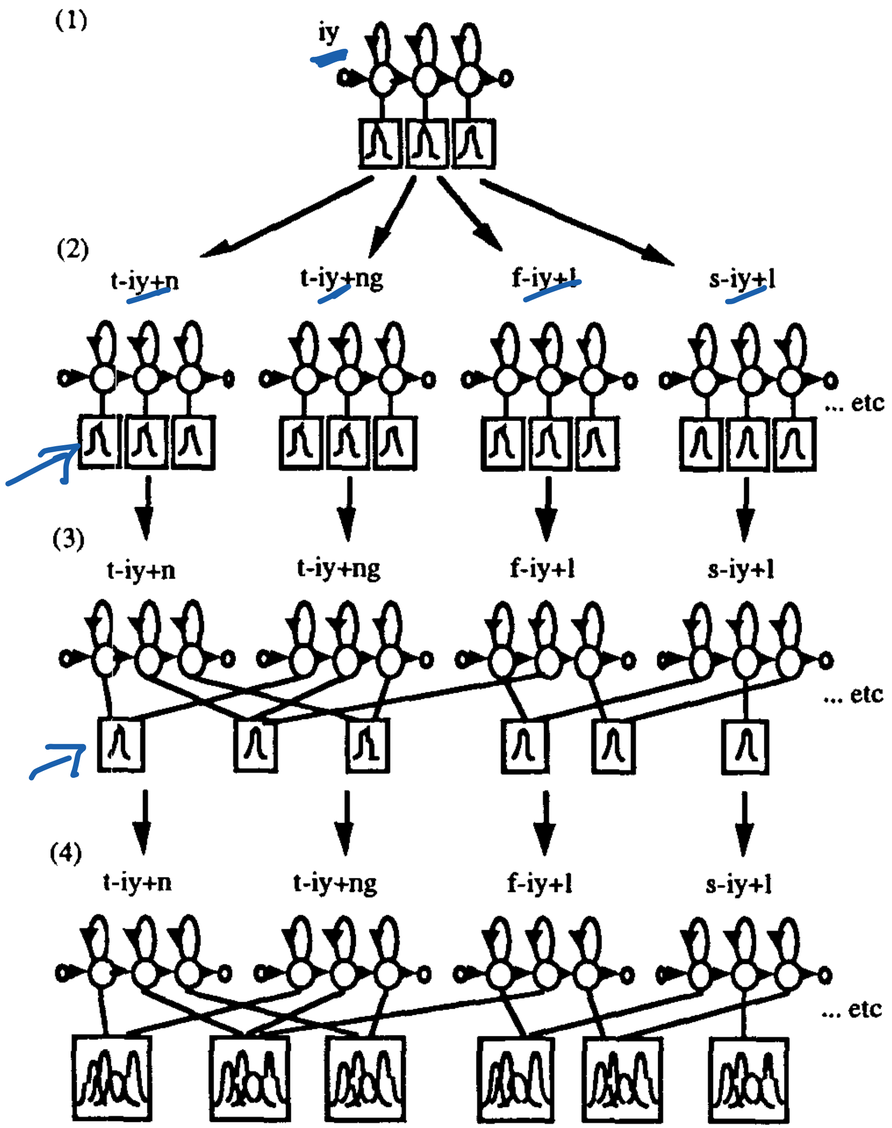
Tied-state HMMs + WFSTs (Live Session)

Lecture 3b



Instructor: Preethi Jyothi, IITB

Tied state HMMs



Four main steps in building a tied state HMM system:

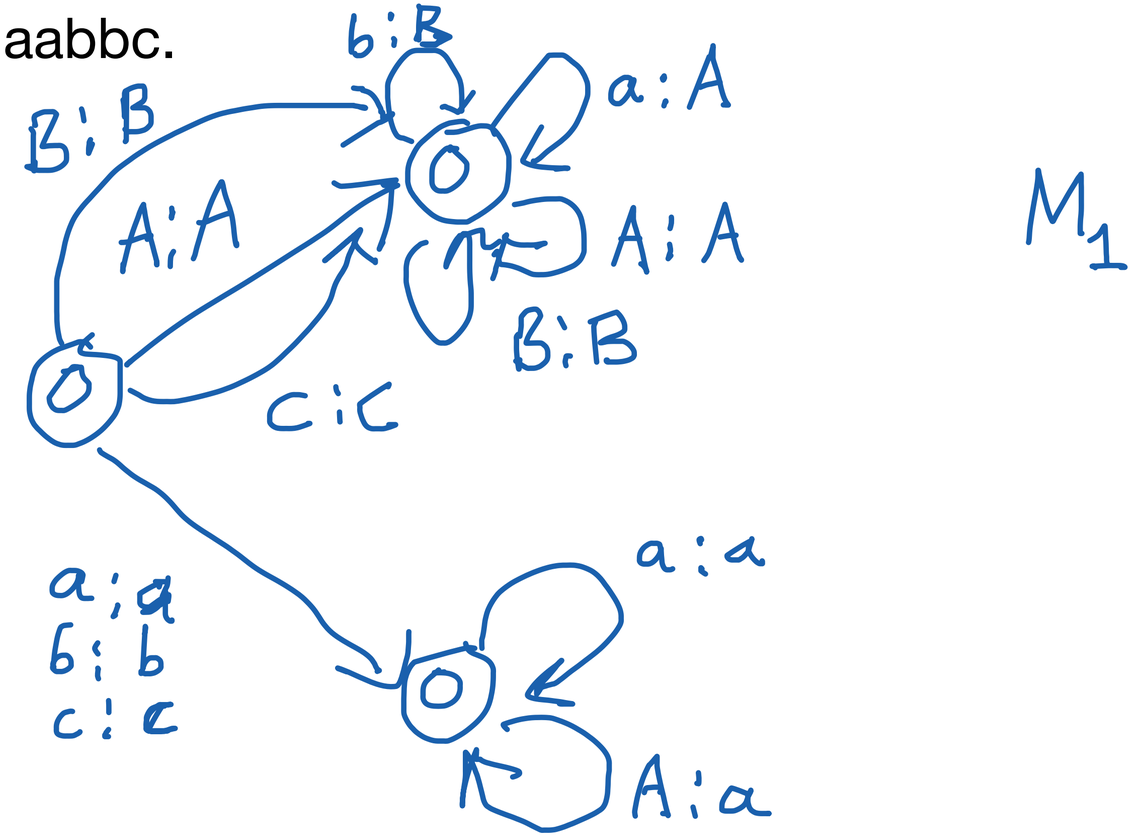
1. Create and train 3-state monophone HMMs with single Gaussian observation probability densities
2. Clone these monophone distributions to initialise a set of untied triphone models. Train them using Baum-Welch estimation. Transition matrix remains common across all triphones of each phone.
3. For all triphones derived from the same monophone, cluster states whose parameters should be tied together.
4. Number of mixture components in each tied state is increased and models re-estimated using BW

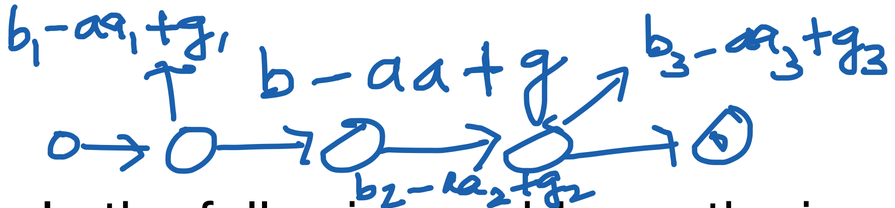
Image from: Young et al., “Tree-based state tying for high accuracy acoustic modeling”, ACL-HLT, 1994

WFST: Problem 1A

In the following problems, the input and output alphabets are both {a, b, c, A, B, C}.

Design a deterministic FST M_1 which takes a string of characters and converts them all to uppercase or lowercase, to match the case of the first character. As an example, the input aABBc will be converted to aabbc.



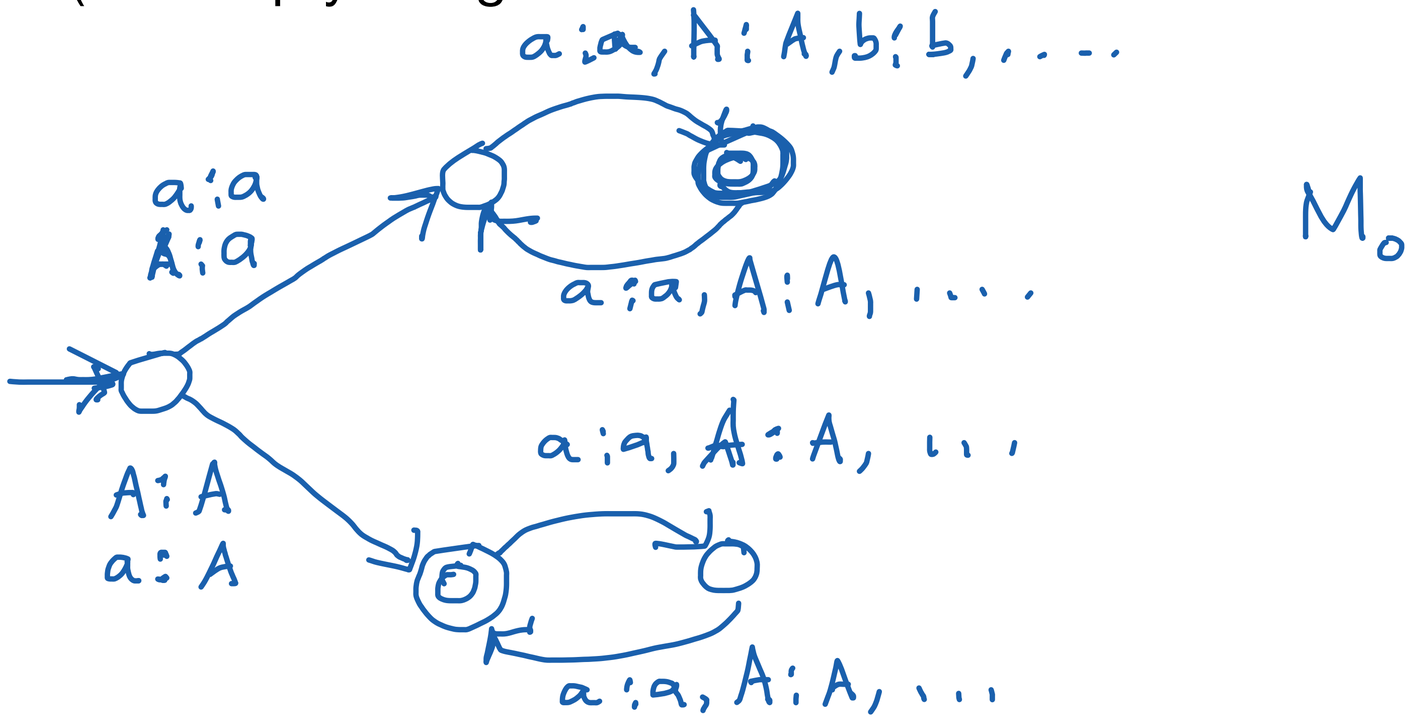


WFST: Problem 1B

$\arg\max_W P(O|W) P(W)$

In the following problems, the input and output alphabets are both {a, b, c, A, B, C}.

Design a non-deterministic FST M_0 which takes a string and outputs it as it is, but changes only the first character as follows: If the string is of (non-zero) even length, then the first character of the output is lowercase and if the string is of odd length, then the first character is uppercase. For example, abC is converted to AbC while input abCB results in the same string as the output. (The empty string should not result in a successful output.)



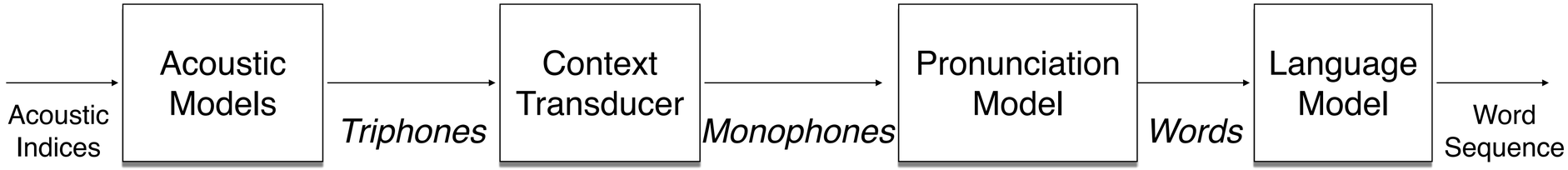
WFST: Problem 1C

Describe, in English, the behaviour of the composed FST $M_0 \circ M_1$.

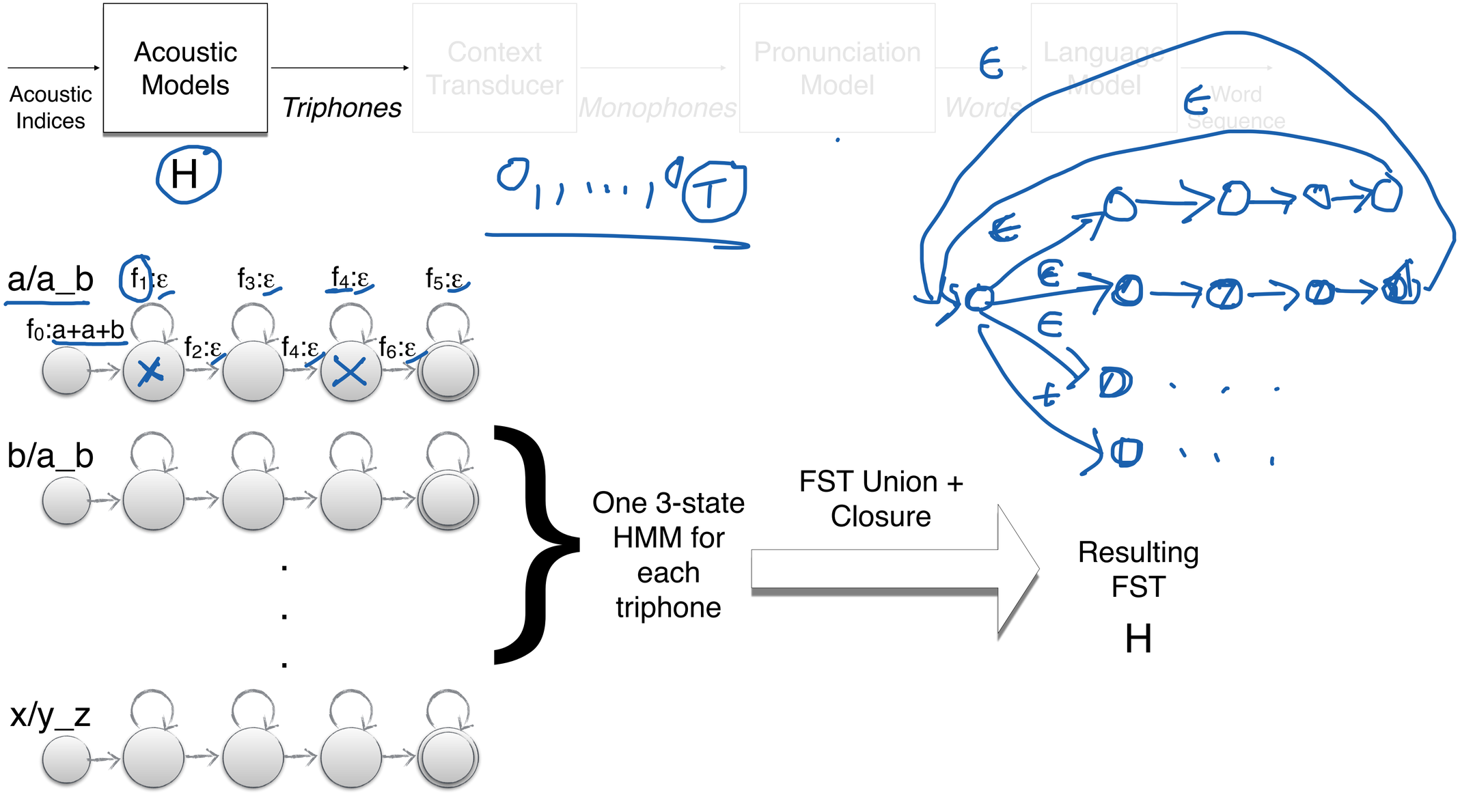
if string is of odd length, convert all chars to uppercase

else, convert all chars to lowercase

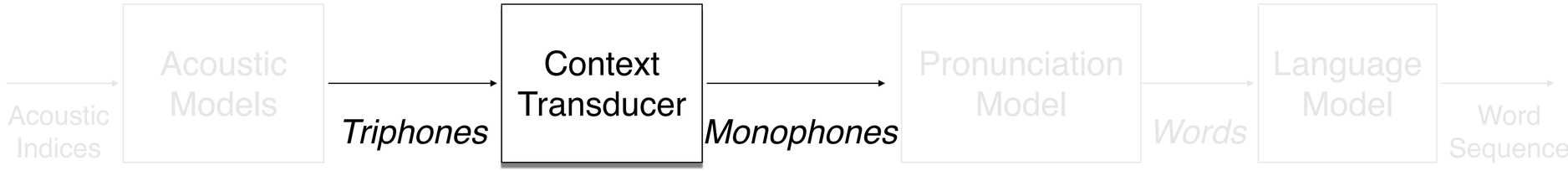
WFST-based ASR System



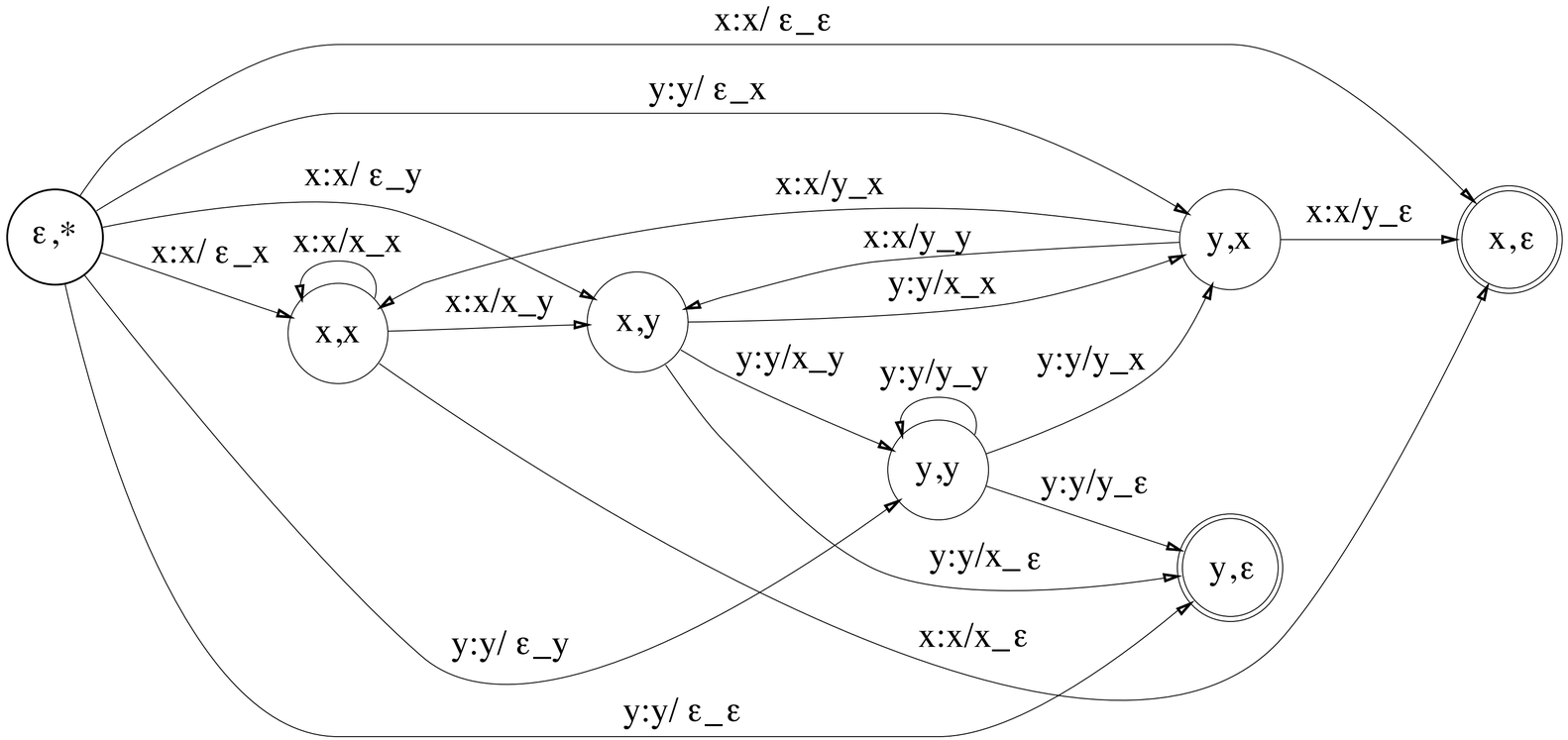
WFST-based ASR System



WFST-based ASR System



C



C^{-1}: Arc labels: “monophone : phone / left-context_right-context”

Figure reproduced from “Weighted Finite State Transducers in Speech Recognition”, Mohri et al., 2002

WFST-based ASR System

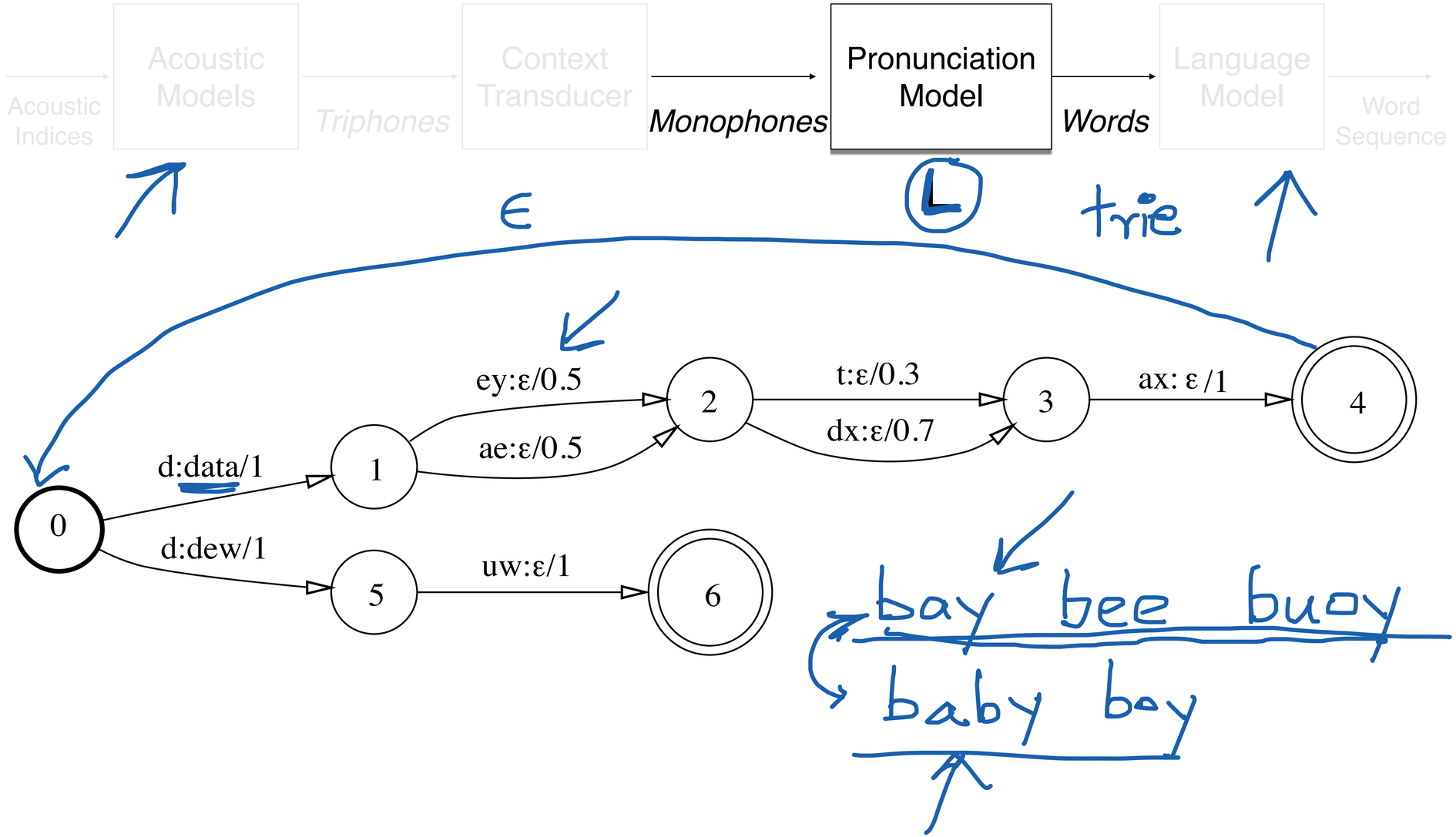
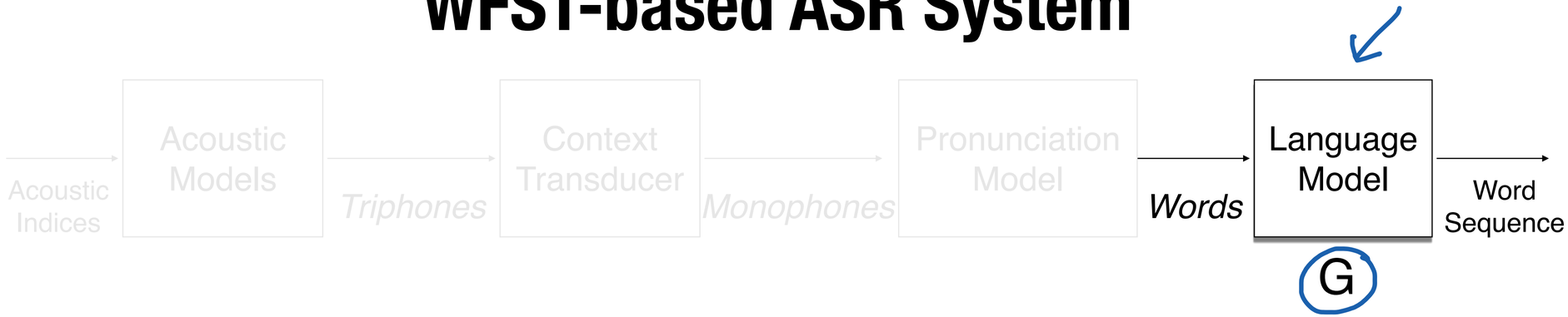
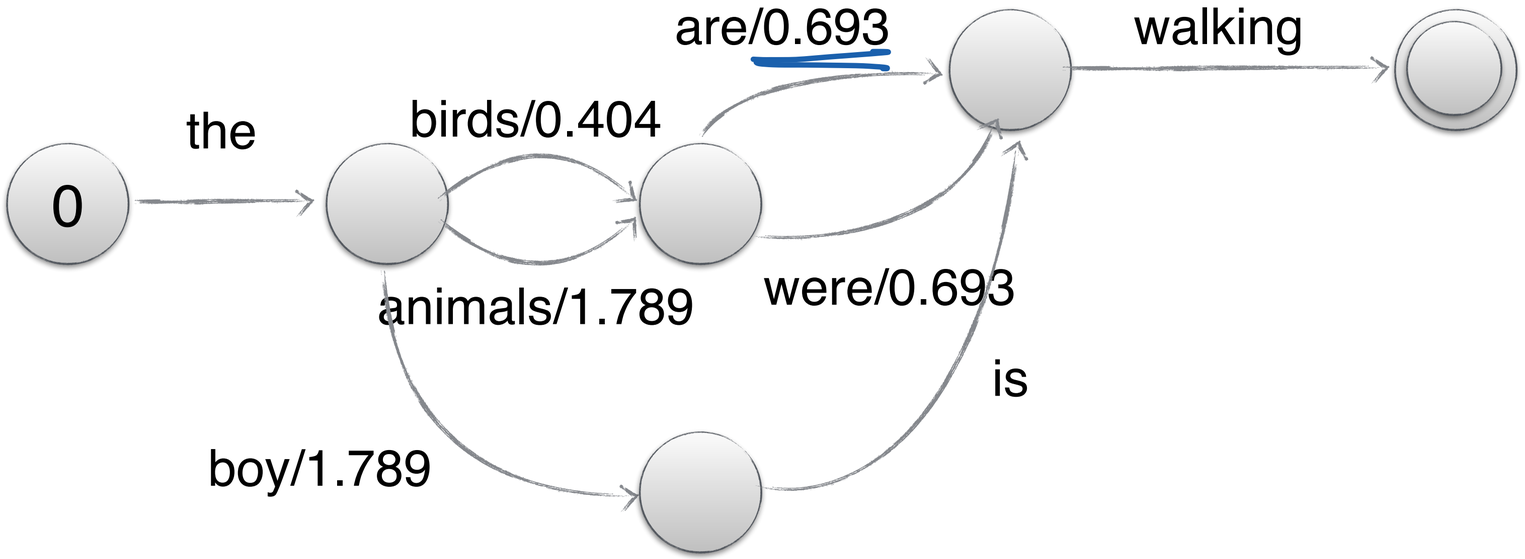


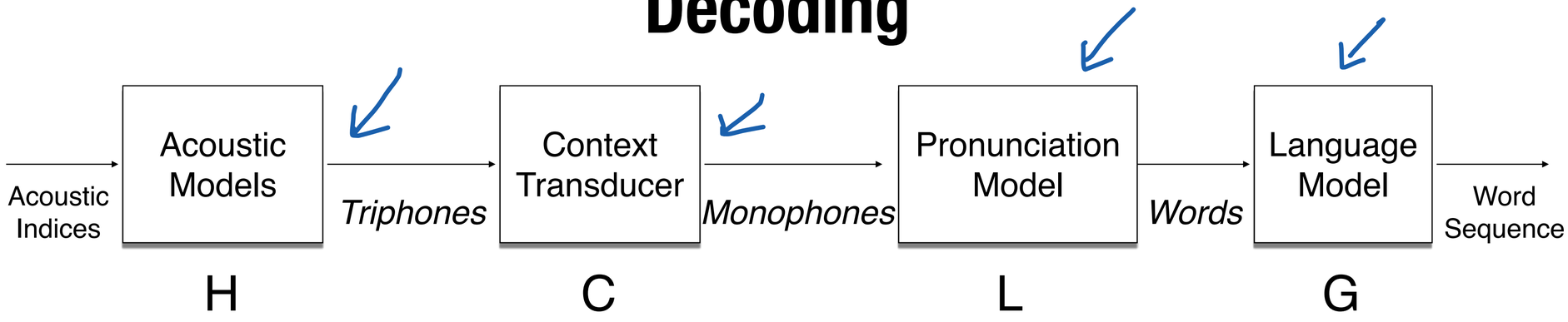
Figure reproduced from "Weighted Finite State Transducers in Speech Recognition", Mohri et al., 2002

WFST-based ASR System





Decoding



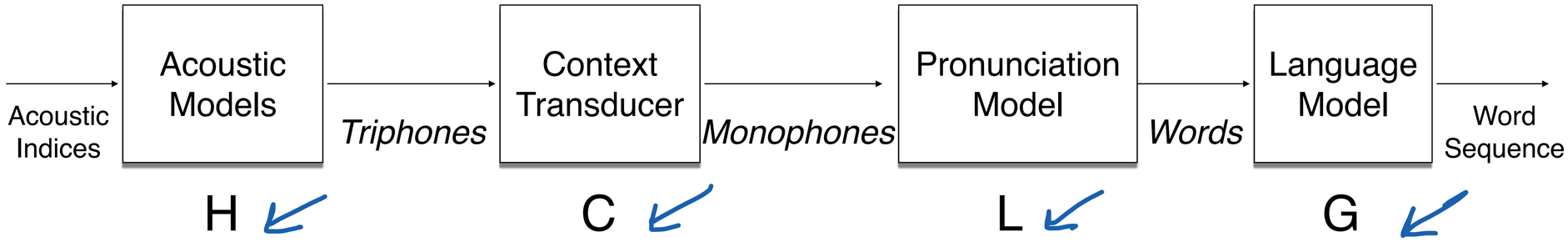
Carefully construct a decoding graph D using optimization algorithms:

$$D = \min(\det(H \circ \det(C \circ \det(L \circ G))))$$

Given a test utterance *O, how do I decode it?*

"Weighted Finite State Transducers in Speech Recognition", Mohri et al., Computer Speech & Language, 2002

Decoding

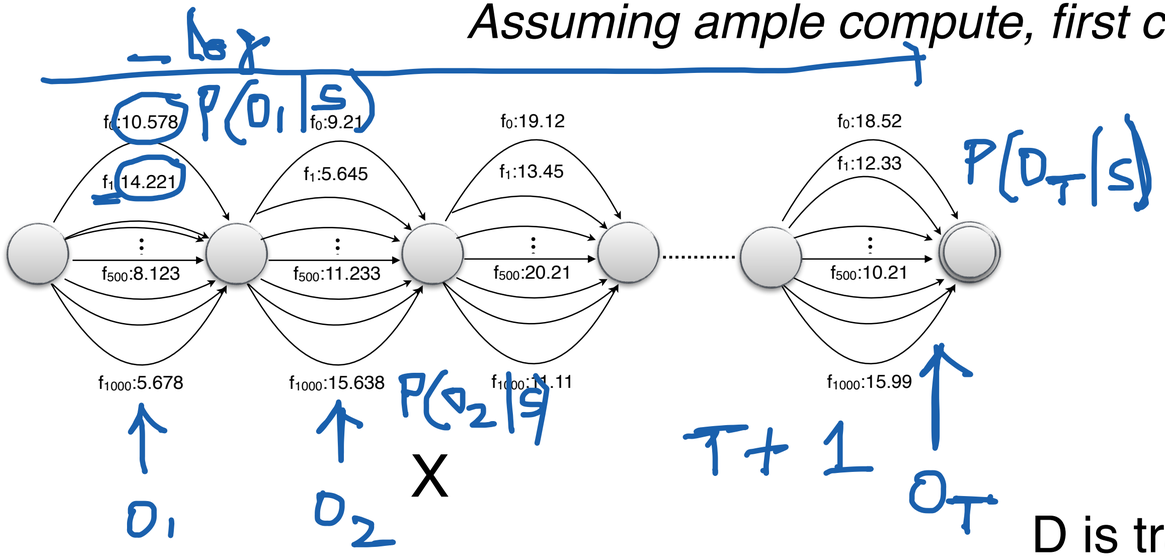


Carefully construct a decoding graph D using optimization algorithms:

$$D = \min(\det(H \circ \det(C \circ \det(L \circ G))))$$

Given a test utterance *O, how do I decode it?*

Assuming ample compute, first construct the following machine X from O.



$$W^* = \arg\min_{W=out[\pi]} X \circ D$$

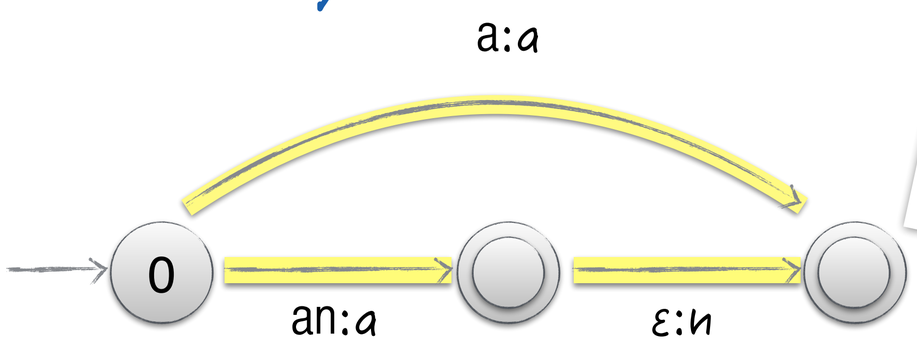
where π is a path in the composed FST
$out[\pi]$ is the output label sequence of π

X is never typically constructed;
D is traversed dynamically using approximate search algorithms
(discussed later in the semester)

Toolkits to work with finite-state machines

- AT&T FSM Library (no longer supported)
  http://www3.cs.stonybrook.edu/~algorith/implement/fsm/implement.shtml
- RWTH FSA Toolkit
  https://www-i6.informatik.rwth-aachen.de/~kanthak/fsa.html
- Carmel
  https://www.isi.edu/licensed-sw/carmel/
- MIT FST Toolkit
  http://people.csail.mit.edu/ilh/fst/
- OpenFST Toolkit (actively supported, used by Kaldi)
  http://www.openfst.org/twiki/bin/view/FST/WebHome

Quick Intro to OpenFst (www.openfst.org)



"0" label is reserved for epsilon

st state
end state
i/p
o/p label
transition

0	1	an	a
1	2	<eps>	n
0	2	a	a
1			
2			

A.txt

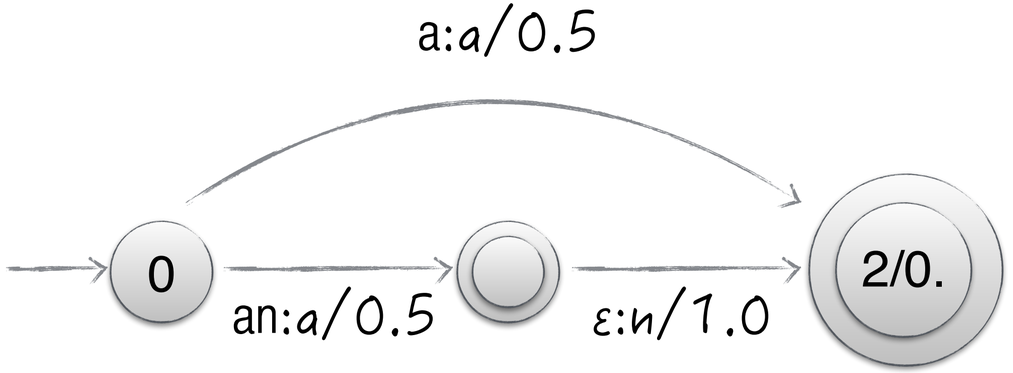
<eps>	0
an	1
a	2

Input alphabet (in.txt)

<eps>	0
a	1
n	2

Output alphabet (out.txt)

Quick Intro to OpenFst (www.openfst.org)



0	1	an	a	0.5
1	2	<eps>	n	1.0
0	2	a	a	0.5
1				
2	0.1			

Compiling & Printing FSTs

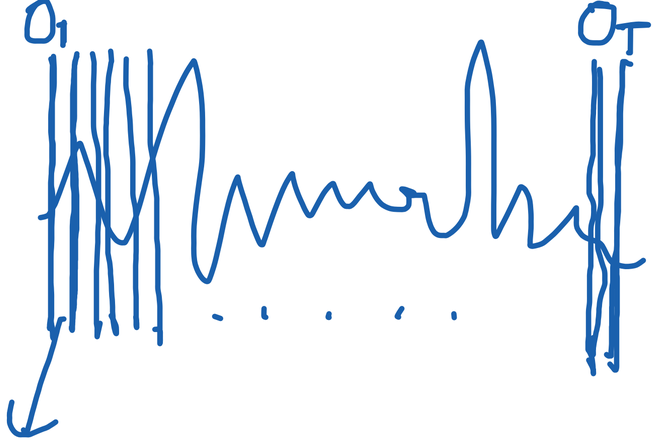
The text FSTs need to be “compiled” into binary objects before further use with OpenFst utilities

- Command used to compile:

```
fstcompile --isymbols=in.txt --osymbols=out.txt A.txt A.fst
```

- Get back the text FST using a print command with the binary file:

```
fstprint --isymbols=in.txt --osymbols=out.txt A.fst A.txt
```



Drawing FSTs

Small FSTs can be visualized easily using the draw tool:

Speech frame

```
fstdraw --isymbols=in.txt --osymbols=out.txt A.fst |
dot -Tpdf > A.pdf
```

≈ 10 - 15 msecs