Learning Weighted Finite-State Transducers

SPFLODD

October 27, 2011

Background

- This lecture is based on a paper by Jason Eisner at ACL 2002, "Parameter Estimation for Probabilistic Finite-State Transducers."
  - This is perhaps the most under-appreciated paper in the past ten years of NLP.
- Full disclosure: Jason Eisner was my Ph.D. advisor.
  - He's one of the smartest people I have ever met.

Finite-State Automaton

Notation	Definition
Q	finite set of states
Σ	finite vocabulary
$q_0 \in Q$	start state
$F \subseteq Q$	set of final states
$\delta : Q \times \Sigma^* \rightarrow 2^Q$	transition function; possible next states given current state and input symbol(s)

Finite-State Automaton (Maybe Better?)

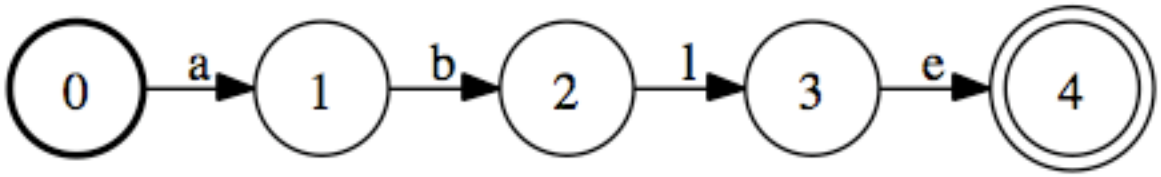
Notation	Definition
Q	finite set of states
Σ	finite vocabulary
$q_0 \in Q$	start state
$F \subseteq Q$	set of final states
$A \subseteq Q \times \Sigma^* \times Q$	set of transitions (source state, symbol sequence, target state)

Finite-State Automaton

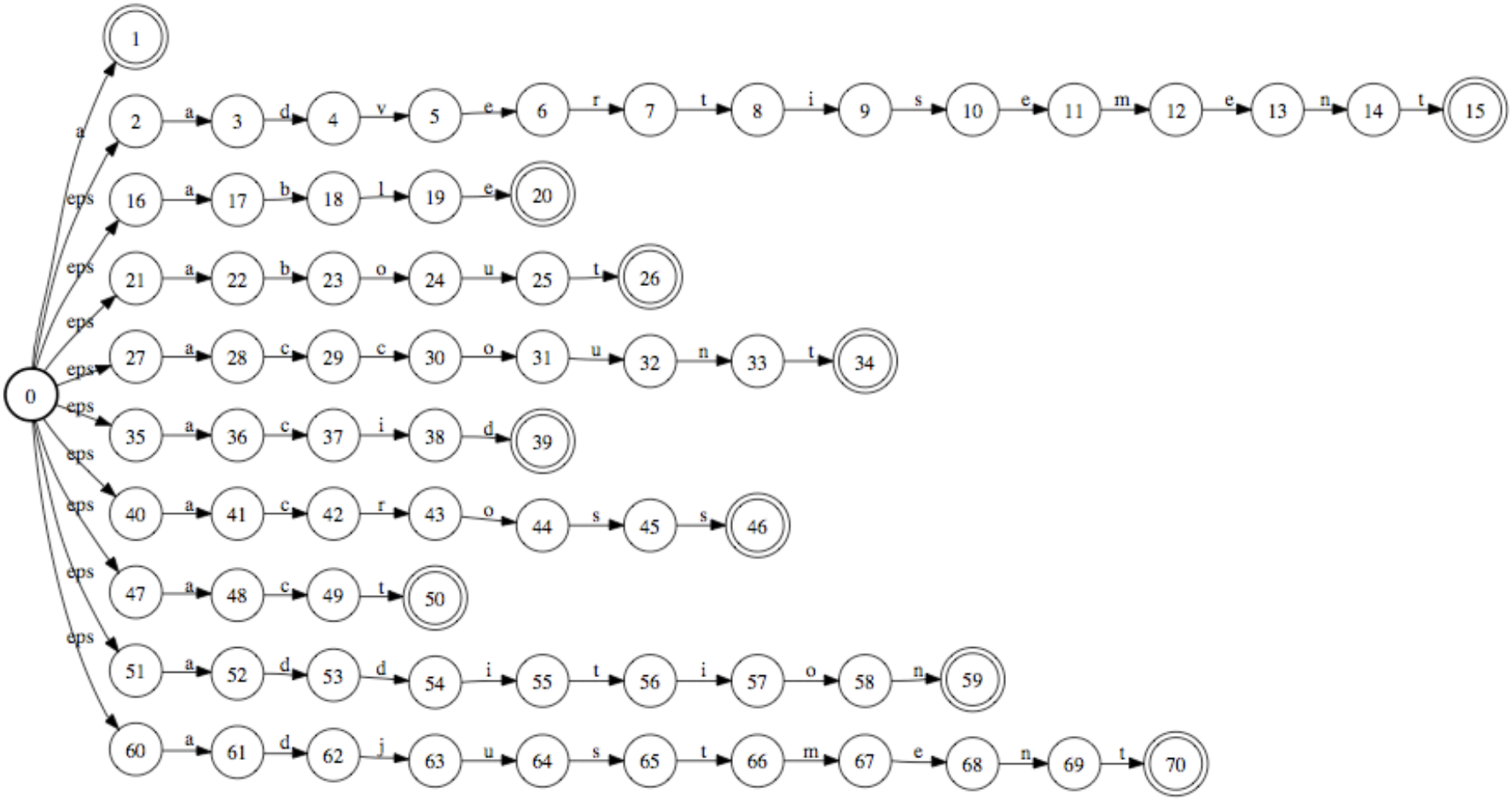
- Automaton that recognizes a **regular** language
  - Key transformations: remove ε-transitions, determinize, minimize
- Implementation of a **regular** expression
- Regular languages are closed under numerous operations
  - Concatenation, union, intersection, Kleene *, difference, reverse, complement, …
- Correspond to regular grammars (type 3 in Chomsky hierarchy)
- Pumping lemma: necessary condition for a language to be regular

FSA as a Dictionary

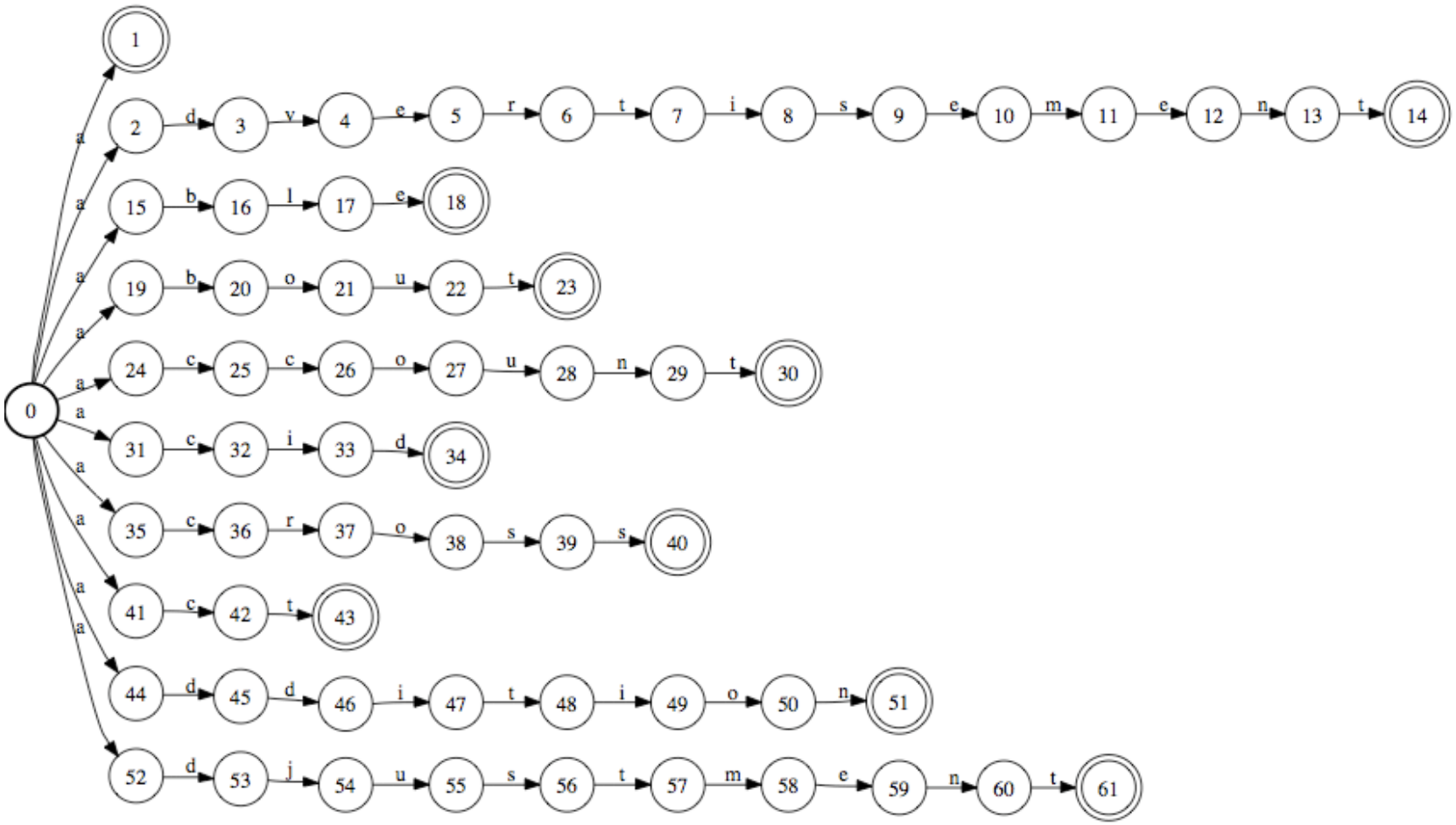
- Example: 850 words in “Basic English”
- Each word is an FSA



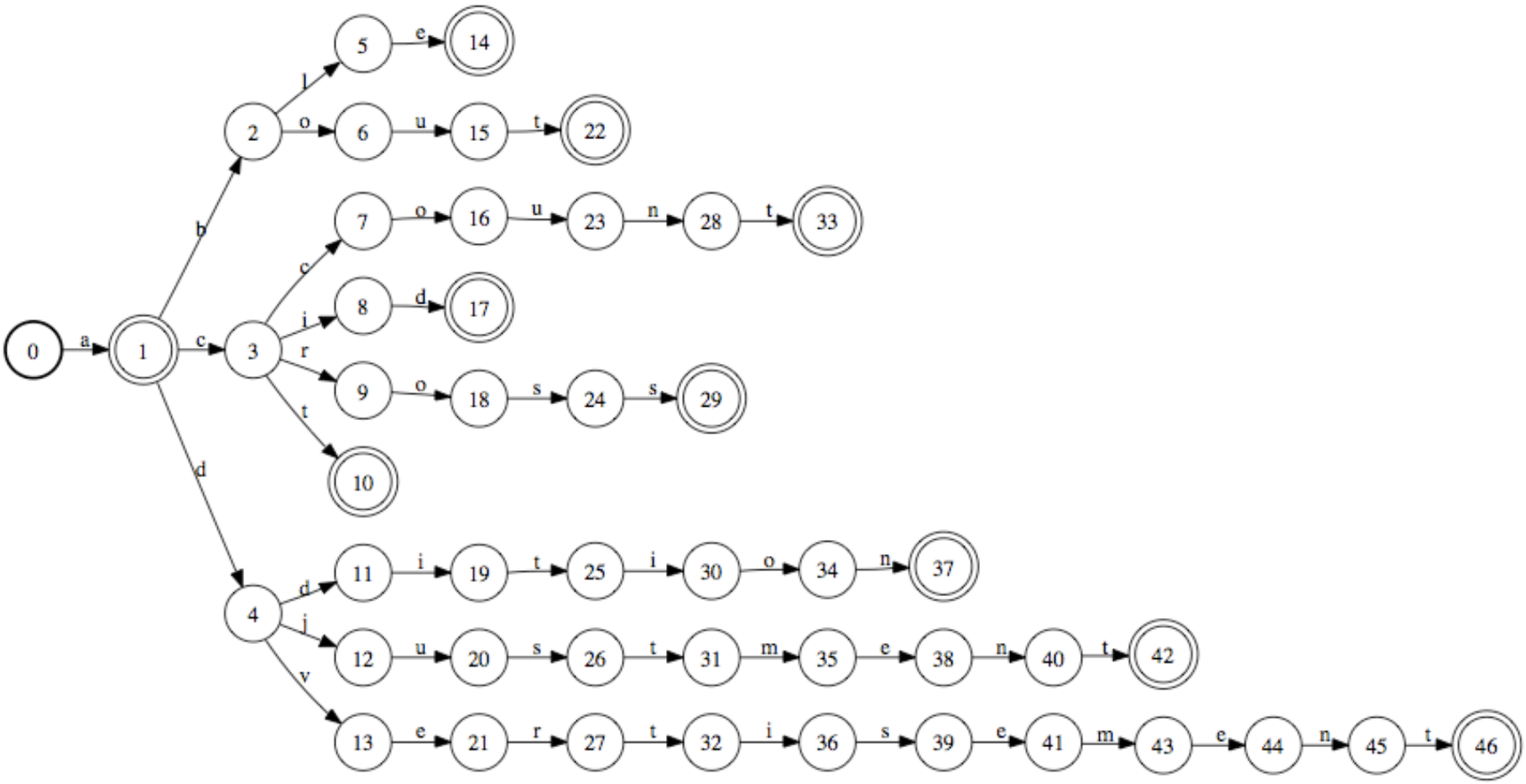
Ten-Word Dictionary



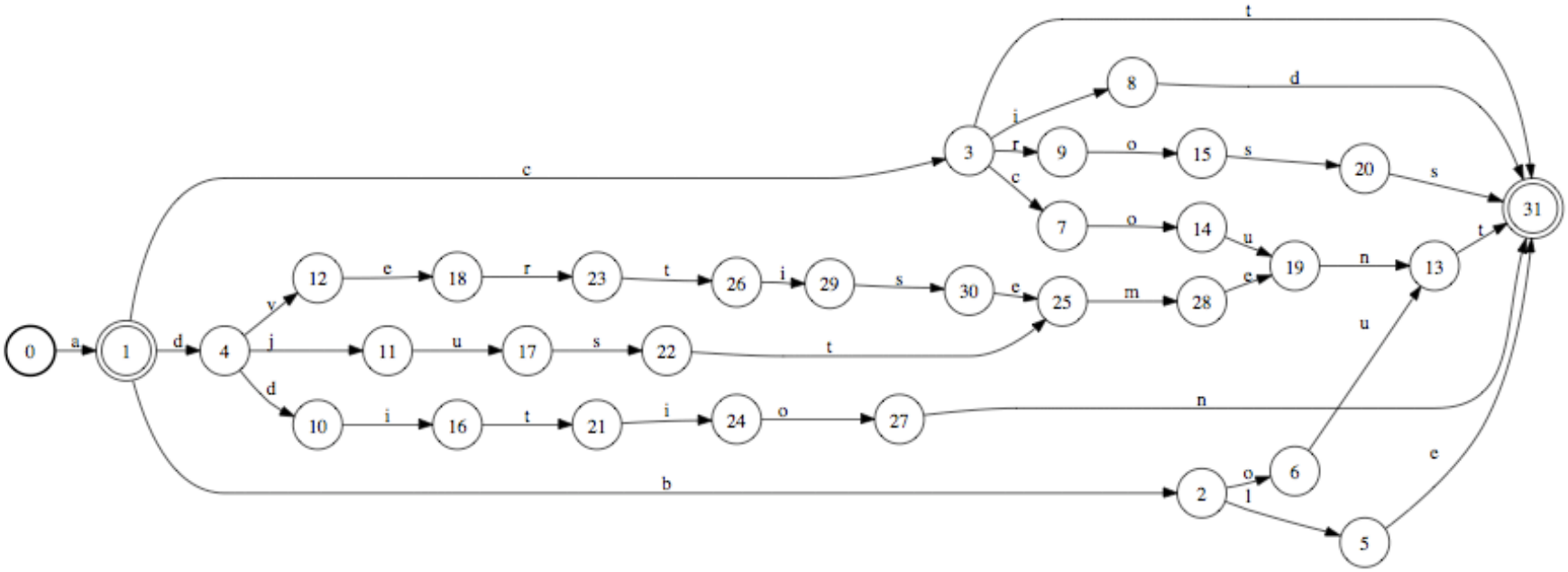
Remove ε-transitions



Determinize



Minimize


0
a
1
d
4
v
12
e
18
r
23
t
26
i
29
s
30
e
25
m
28
e
19
n
13
t
31
j
11
u
17
s
22
t
d
10
i
16
t
21
i
24
o
27
n
c
3
t
8
d
j
r
9
o
15
s
20
s
c
7
o
14
u
b
2
o
l
6
5
u
e

Full 850-Word Dictionary

	states	*final states*	*arcs*
Union	5303	850	5302
Remove ε-transitions	4454	850	4453
Determinize	2609	848	2608
Minimize	744	42	1535

Generalizations

- Finite-state recognizer is a function from $\Sigma^* \rightarrow \{0,1\}$
  - Meaning: fsa(s) = 1 $\Leftrightarrow$ s is in the language
- Other **rational relations** …
  - Finite-state transducer: $\Sigma^* \rightarrow \Delta^*$
  - Weighted FSA: $\Sigma^* \rightarrow \mathbb{R}$
  - Weighted FST: $\Sigma^* \rightarrow \Delta^* \times \mathbb{R}$
- WFSAs and WFSTs can be considered **probabilistic** (but don't have to be)

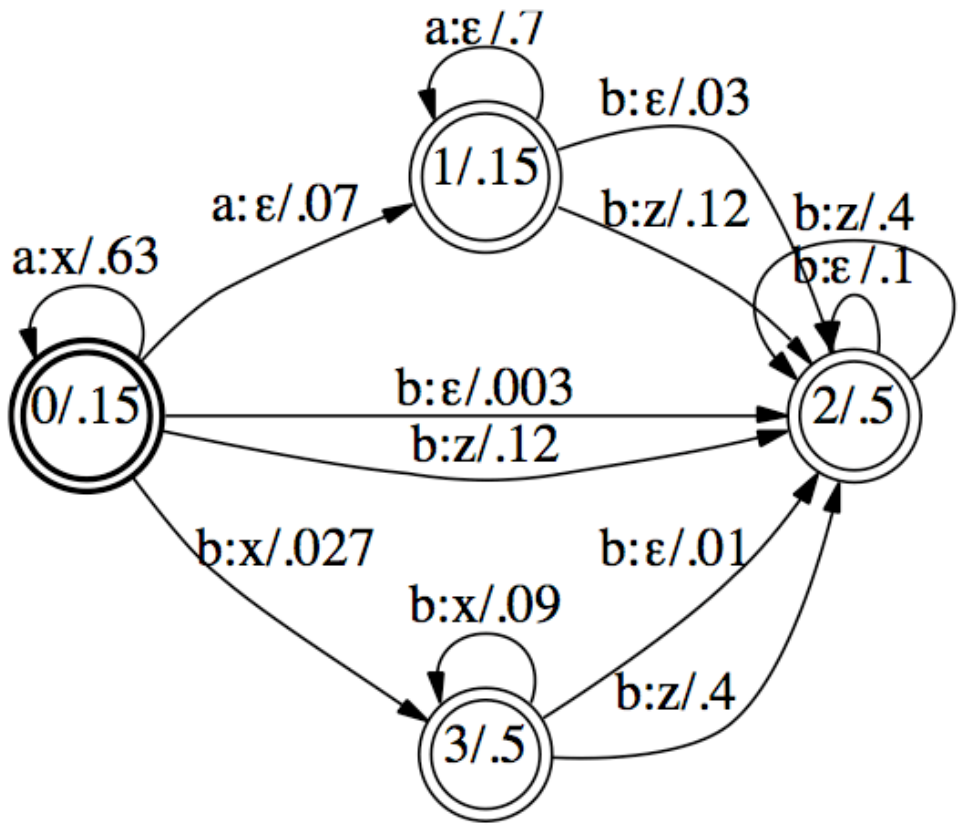
Relations on Strings

- A **relation** is a set of (input, output) pairs.
  - More general than *functions* because you can represent ambiguity and optionality!
  - For standard FSAs, think of input = output.
- Rational relations are a special kind of relation with a wide range of **closure** properties.
- Rational relations can be understood as a declarative programming paradigm:
  - source code is a regular expression
  - object code is a 2-tape automaton called a **finite-state transducer (FST)**
  - optimization is accomplished by determinization and minimization
  - supports nondeterminism, parallel processing over infinite sets of input strings, reverse computation from outputs to inputs

Finite-State Automata and Transducers

FSA	FST	Definition
Q		finite set of states
Σ		finite (input) vocabulary
	Δ	finite output vocabulary
$q_0 \in Q$		start state
$F \subseteq Q$		set of final states
$\delta : Q \times \Sigma^* \rightarrow 2^Q$		transition function; possible next states given current state and input symbol(s)
	$\delta : Q \times \Sigma^* \times \Delta^* \rightarrow 2^Q$	… and output symbol(s)

Eisner's Running Example

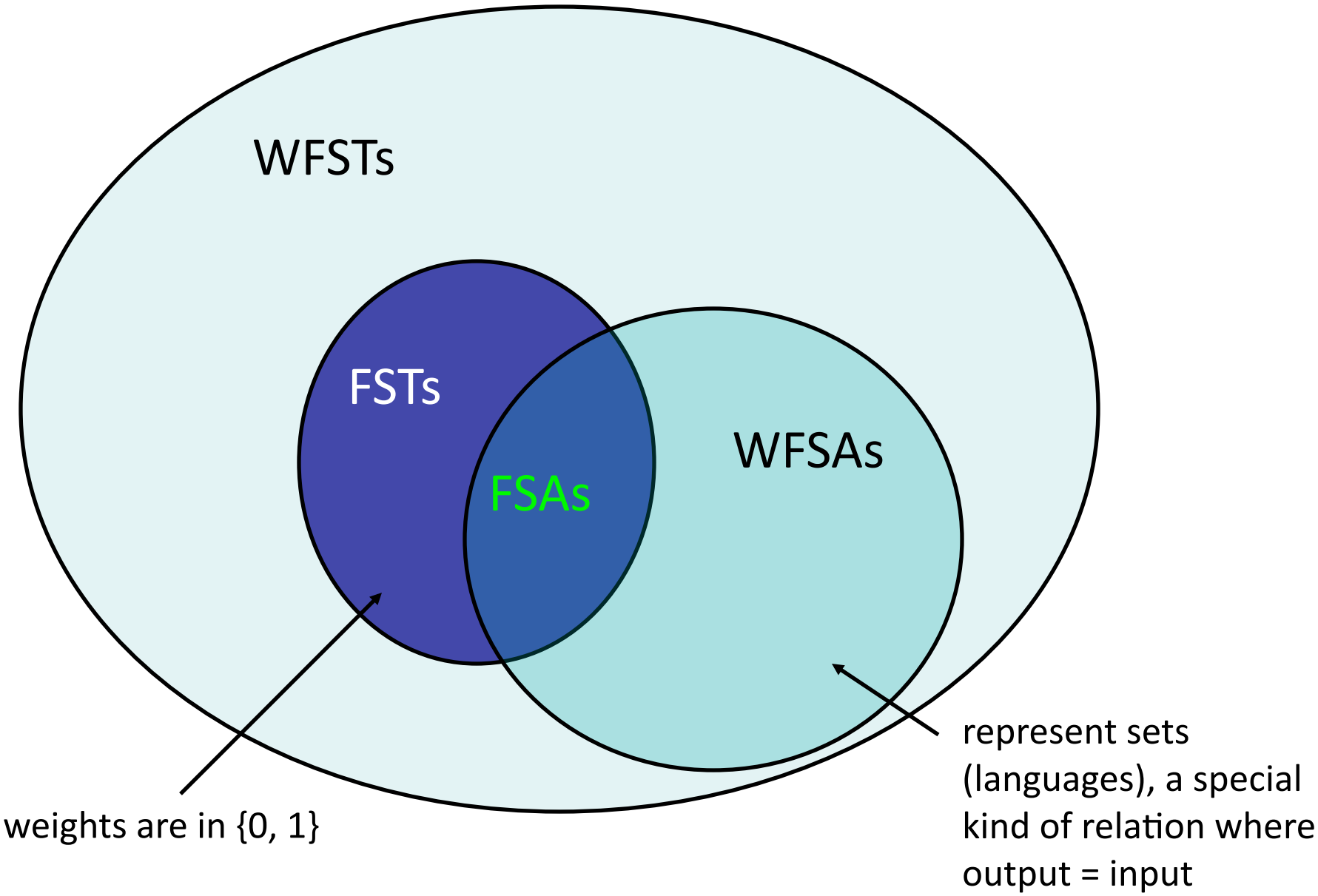


Weighted Relations

- Assign **scores** to (input, output) pairs.
  - Sometimes interpreted as p(input, output)
  - Sometimes interpreted as p(output | input)
  - Sometimes neither
- This idea unifies many NLP approaches:
  - sequence labeling
  - chunking
  - normalization
  - segmentation
  - alignment
  - speech recognition (Pereira and Riley, 1997)
  - machine translation (Knight and Al-Onaizan, 1998)

Weighted FSTs

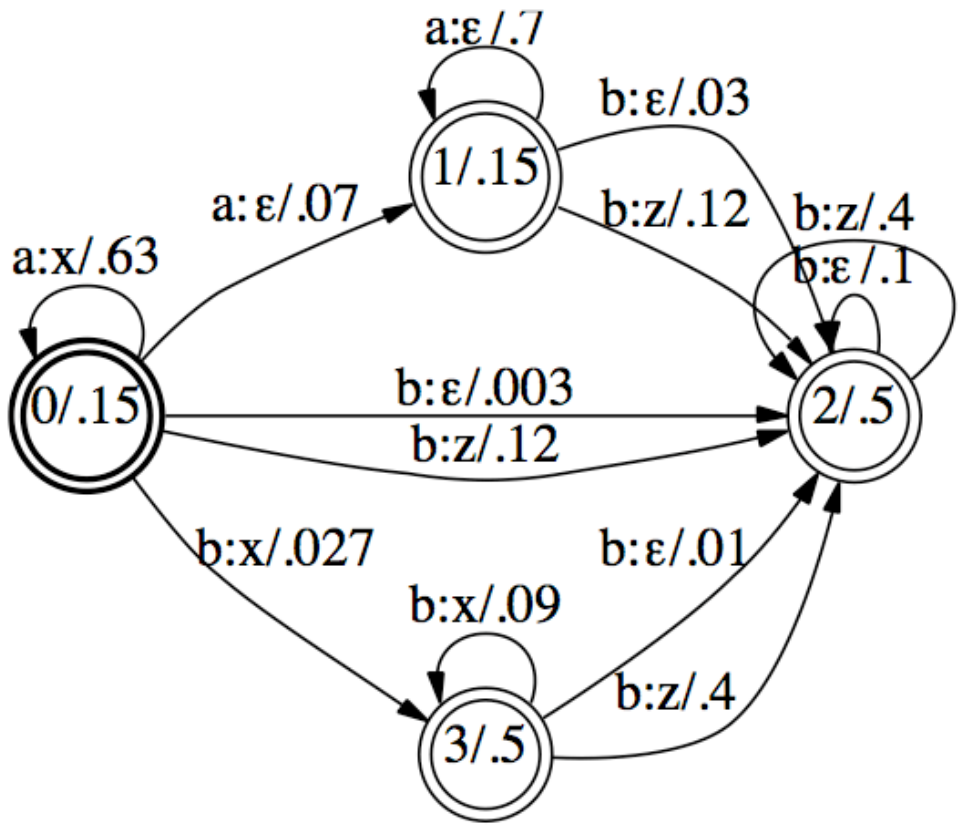
FST	Weights	Definition
Q		finite set of states
Σ		finite input vocabulary
Δ		finite output vocabulary
$q_0 \in Q$		start state
$F \subseteq Q$	stop weights	set of final states
$\delta : Q \times \Sigma^* \times \Delta^* \rightarrow 2^Q$	arc weights	transition function; possible next states given current state and input symbol(s) and output symbol(s)


WFSTs
FSTs
FSAs
WFSAs
weights are in {0, 1}
represent sets (languages), a special kind of relation where output = input

Weights and Scores

- Weights are assigned to transitions and to ending a path in each state.
- **Score** of a path is the product of the transition weights and the stop weight.
- “Zero” means the same thing as “impossible.”

Eisner's Running Example



Two paths for (aabb, xz):

$$0 \xrightarrow{\mathtt{a}:\mathtt{x}/.63} 0 \xrightarrow{\mathtt{a}:\epsilon/.07} 1 \xrightarrow{\mathtt{b}:\epsilon/.03} 2 \xrightarrow{\mathtt{b}:\mathtt{z}/.4} 2/.5$$

path score = 0.0002646

$$0 \xrightarrow{\mathtt{a}:\mathtt{x}/.63} 0 \xrightarrow{\mathtt{a}:\epsilon/.07} 1 \xrightarrow{\mathtt{b}:\mathtt{z}/.12} 2 \xrightarrow{\mathtt{b}:\epsilon/.1} 2/.5$$

path score = 0.0002646

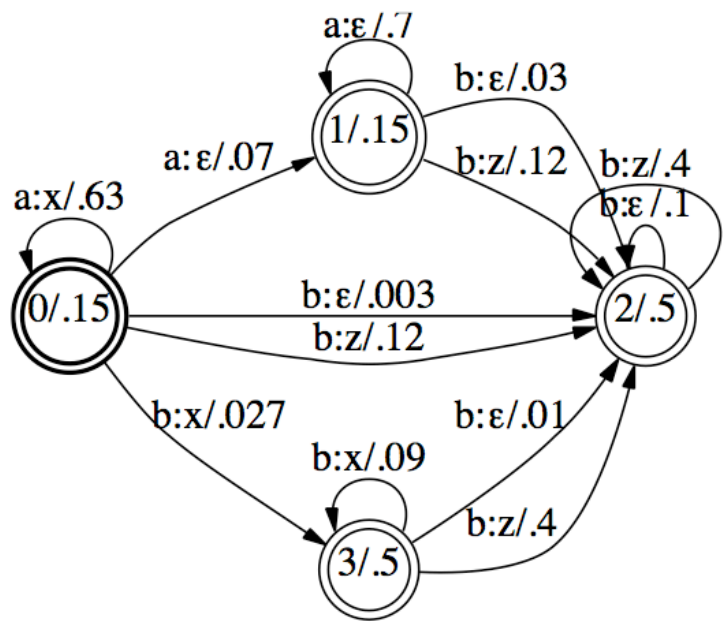
score of (aabb, xz) = 0.0005296

Weighted Relations and Probabilities

- Let f(x, y) be the function corresponding to a WFST's assignment of a score to the (input, output) pair (x, y).
  - If f sums to one over all (x, y), then it is a joint distribution.
  - If f sums to one over all y, for each x, then it is a conditional distribution.

Parameterizing the WFST

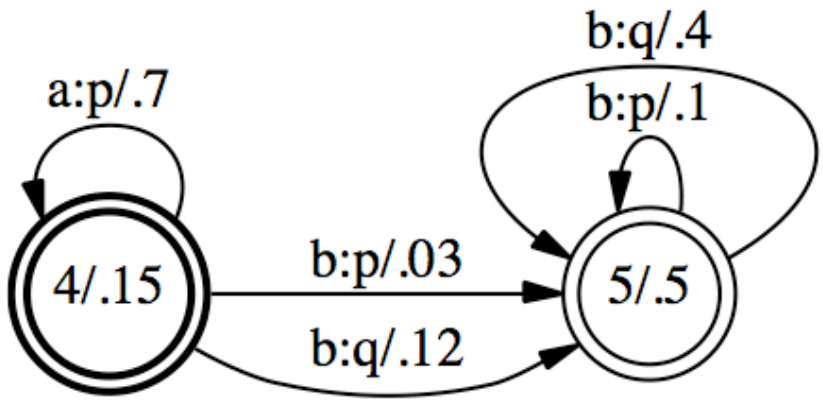
- Option 1: every transition or stop gets a parameter.
  - Option 1A: make sure competing choices (transitions from q and stopping in q) sum to 1.

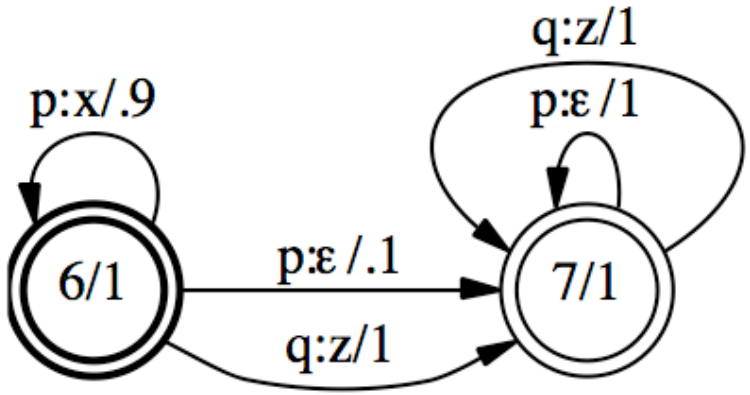


13 free parameters

Parameterizing the WFST

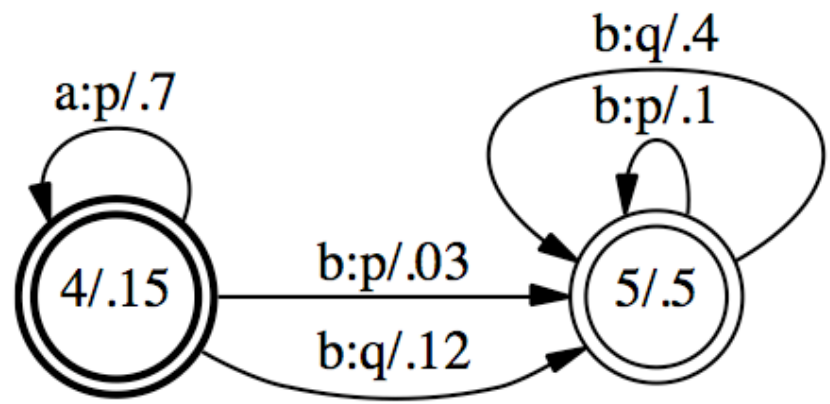
- Suppose our WFST was built by **composing** two simpler WFSTs.



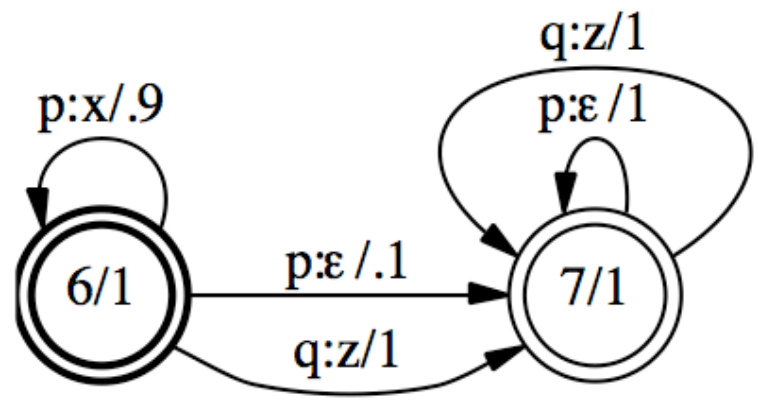


WFST Composition

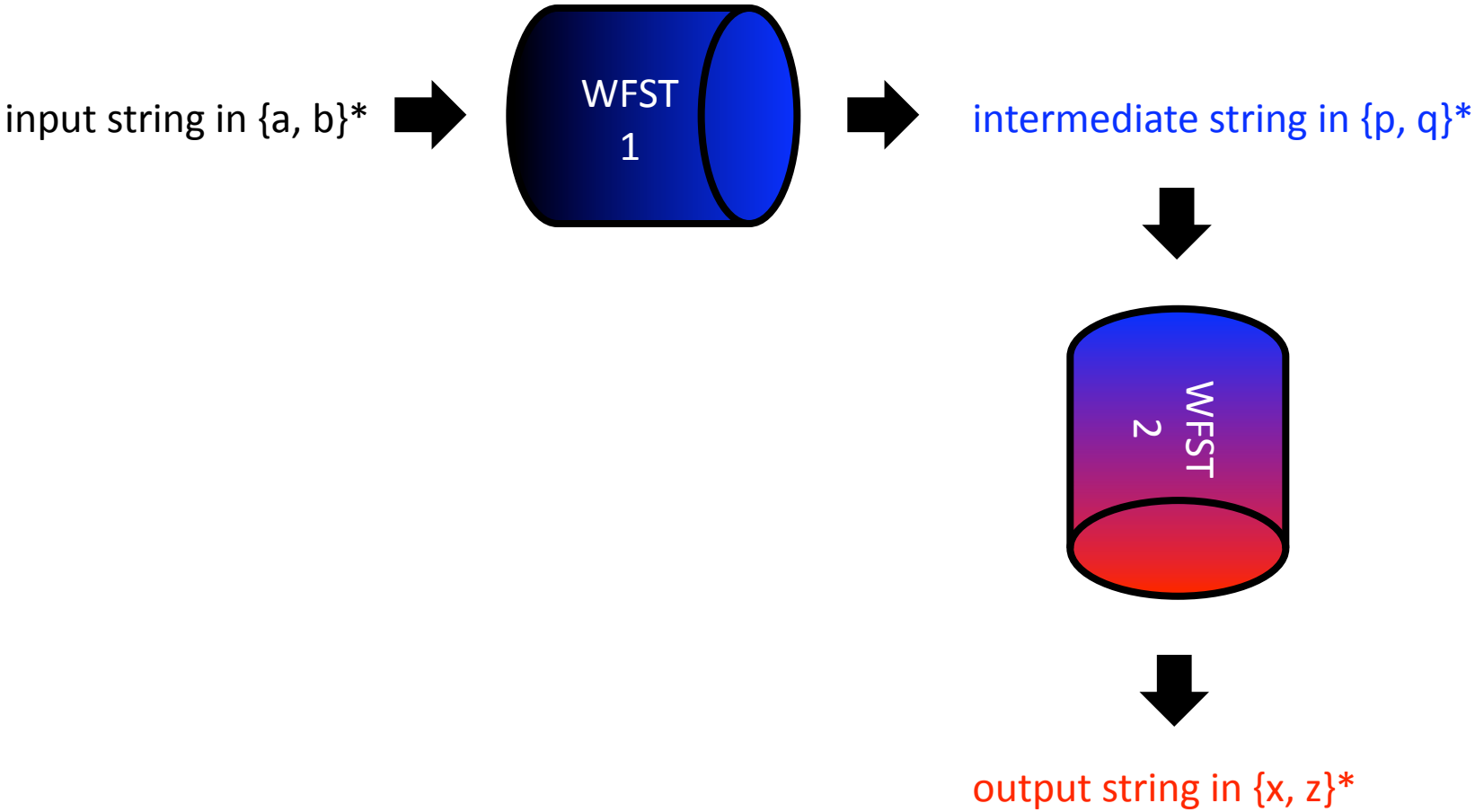
Σ = {a, b}
Δ = {p, q}



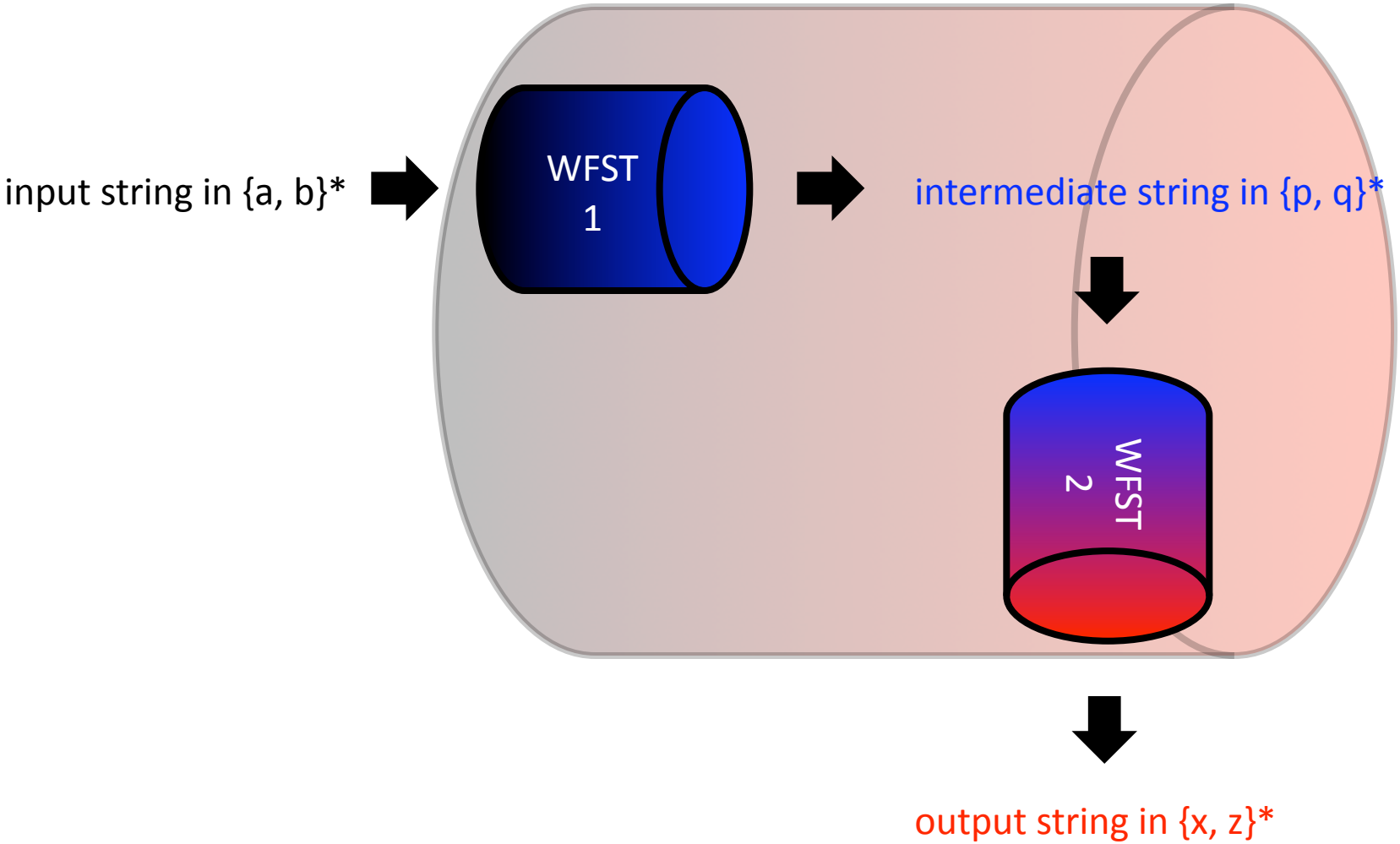
Σ = {p, q}
Δ = {x, z}



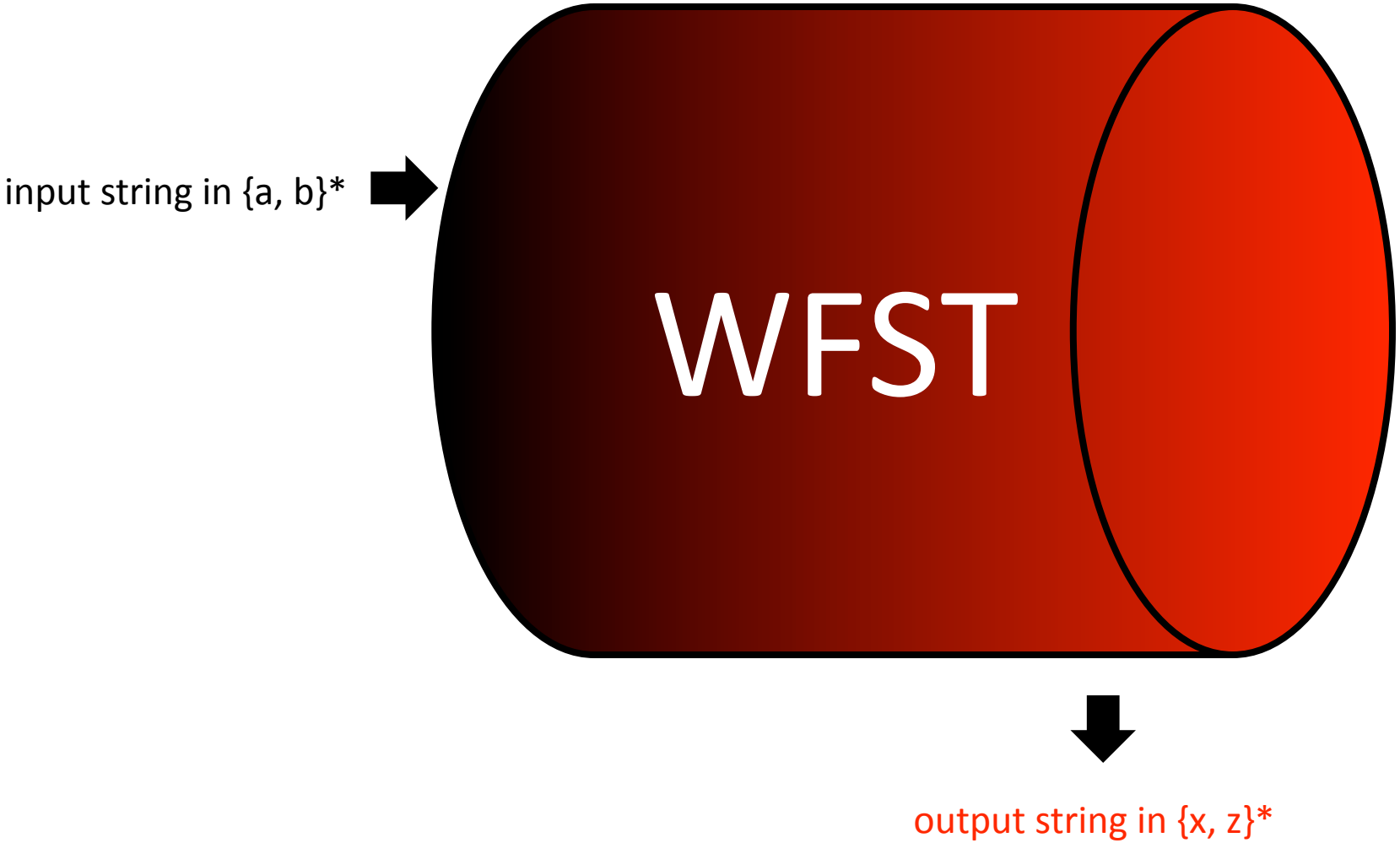
WFST Composition



WFST Composition



WFST Composition



WFST Composition

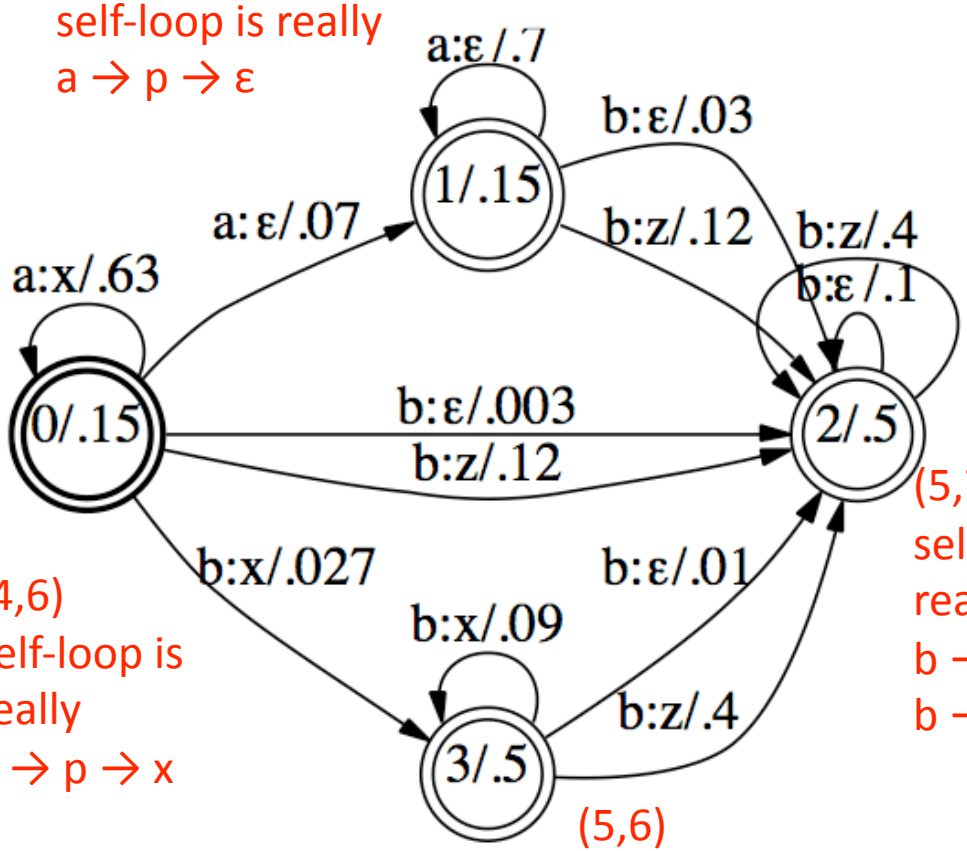
- Let f and g define the weighted relations for two WFSTs such that f's output alphabet and g's input alphabet are the same.

  Then:
  $f \circ g\ (x, z) = \sum_y f(x, y) \times g(y, z)$

  - Either f or g or both can be a set (instead of a relation).
  - Either f or g can be unweighted (scores are 0 or 1).
  - If both are unweighted sets (FSAs) then this is **intersection**.
- If f is a joint distribution $p(x, y)$ and g is a conditional distribution $p(z \mid y)$, we now have a probabilistic model over three string random variables.

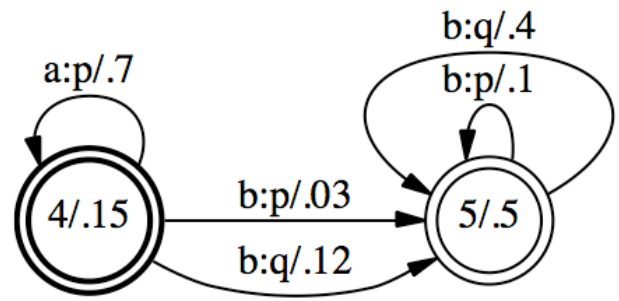
WFST Composition

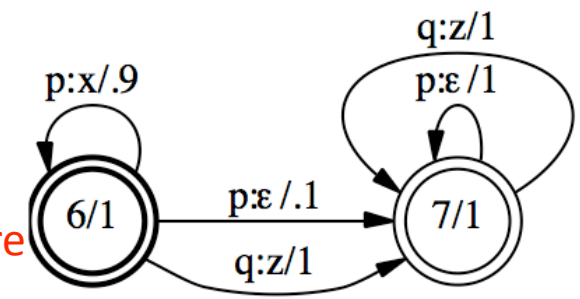
(4,7)
self-loop is really
a → p → ε



(4,6)
self-loop is
really
a → p → x

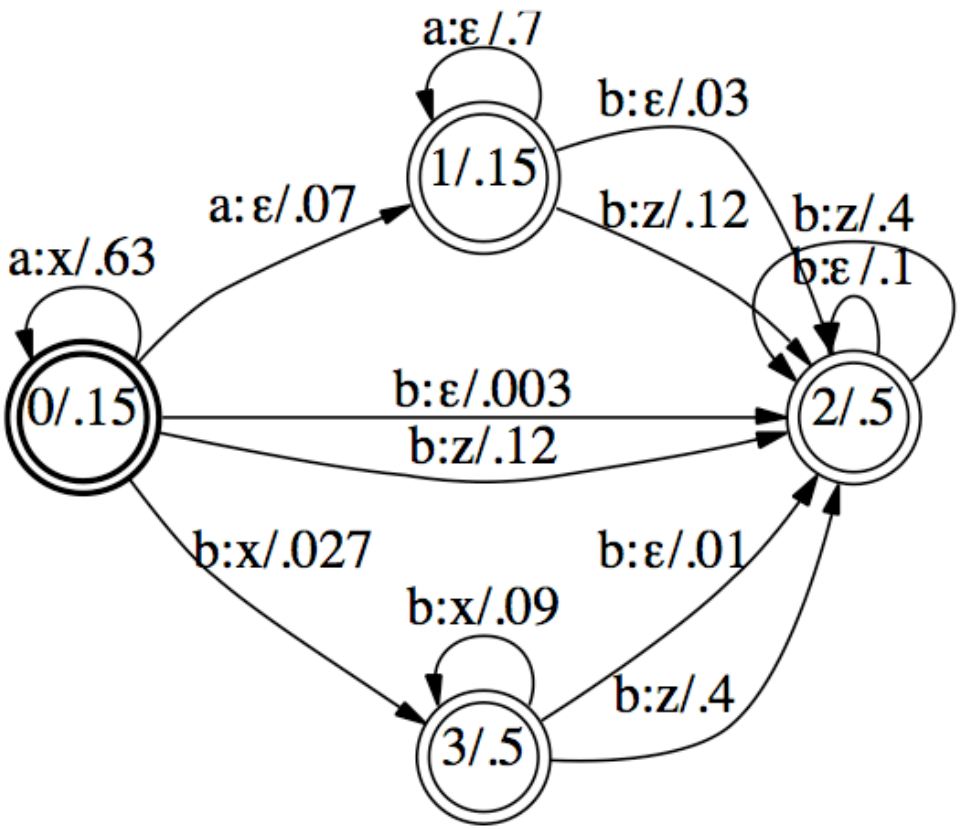
(5,6)
self-loop is really
b → {p, q} → x

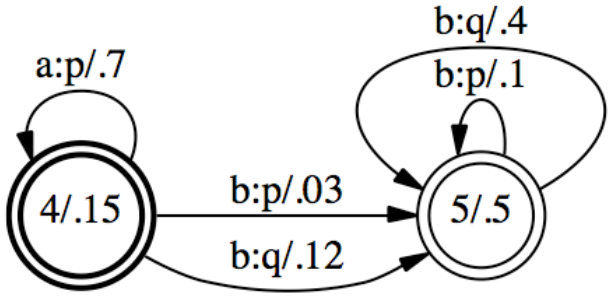




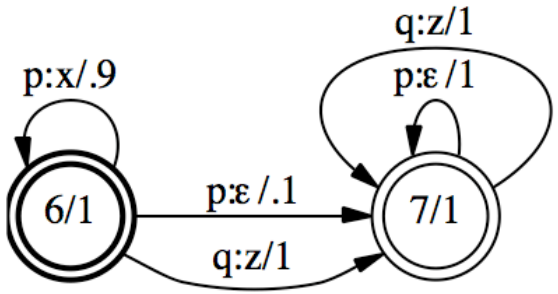
(5,7)
self-loops are
really
b → p → ε and
b → q → z

WFST Composition





6 free parameters



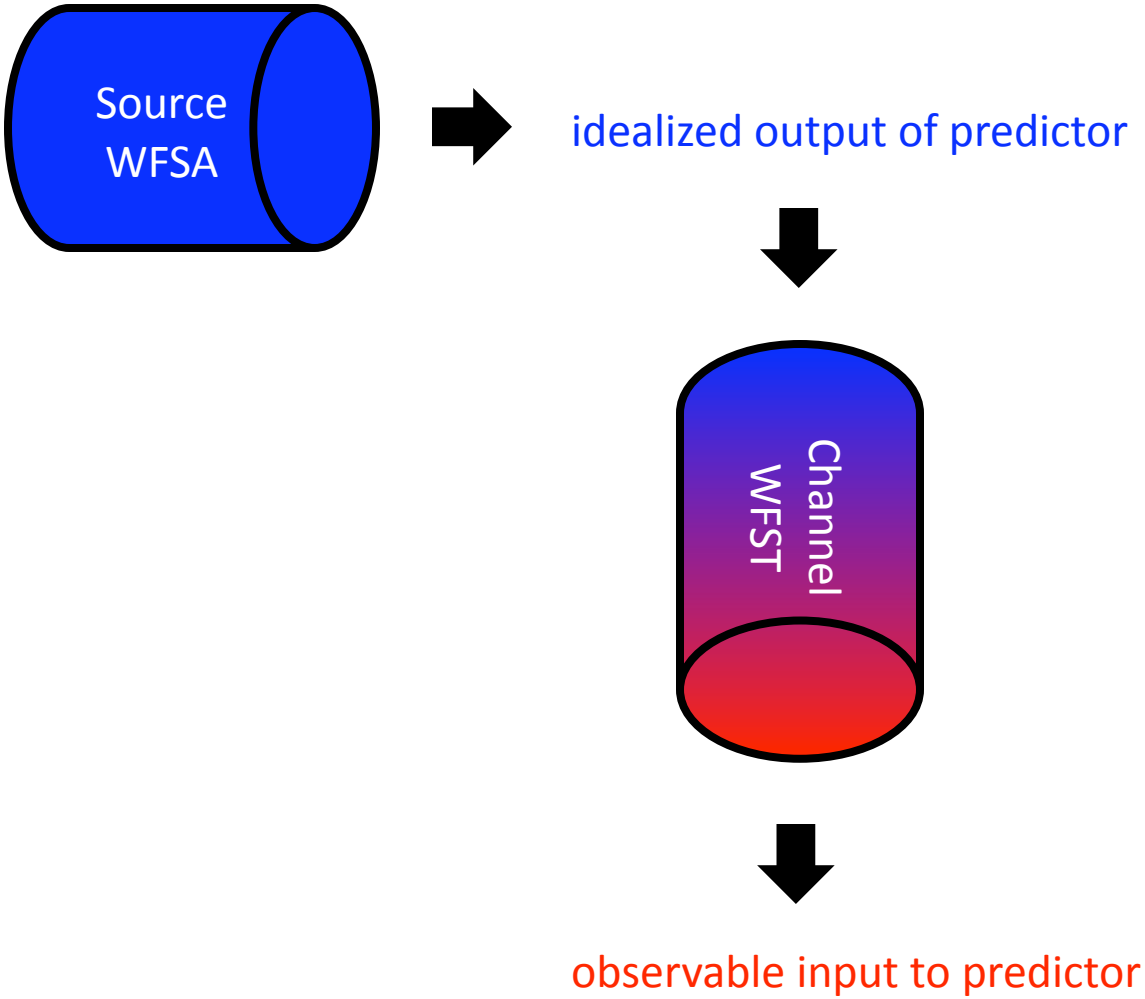
1 free parameter

6 + 1 = 7 free parameters

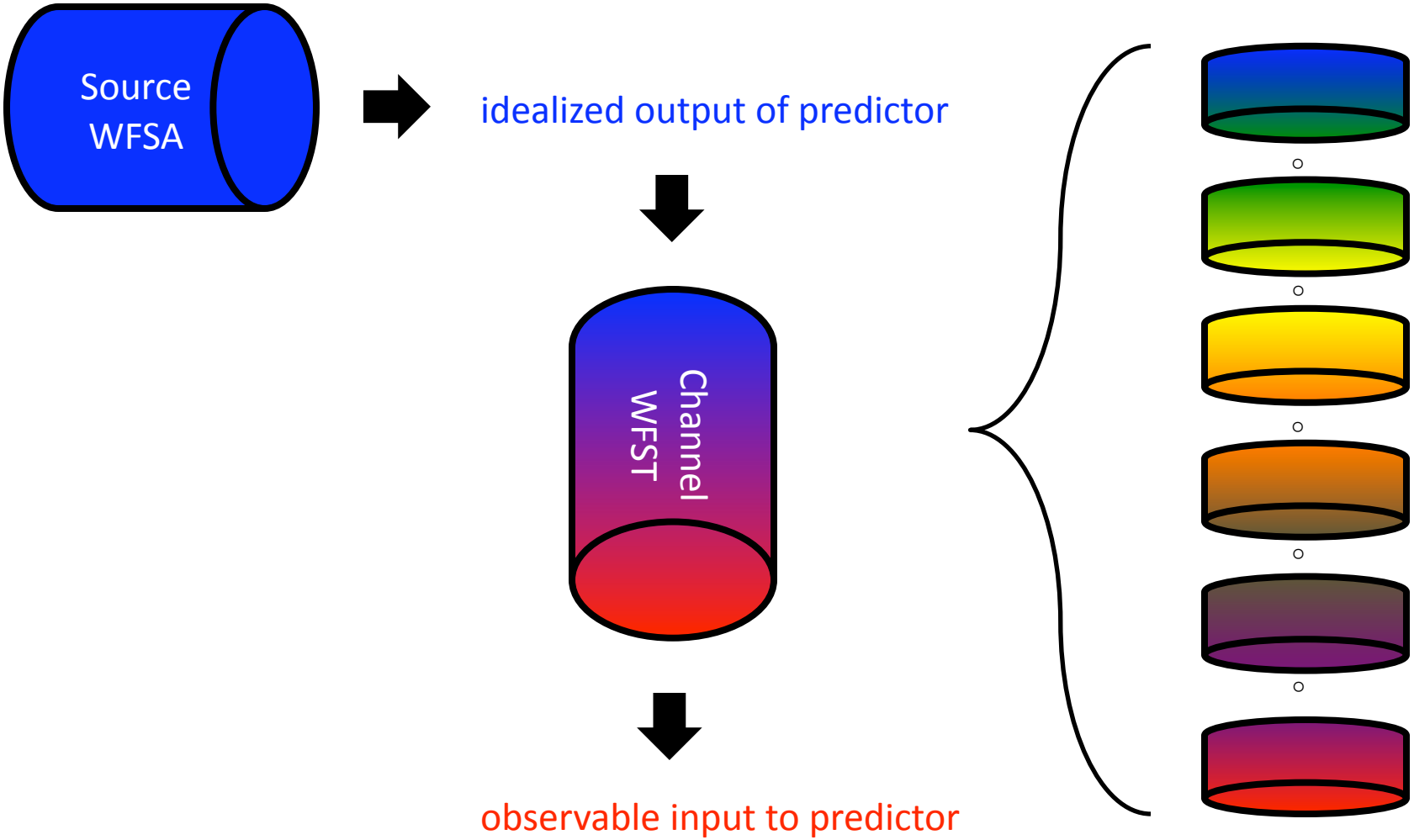
Aside 1

- Eisner suggests another way to write down weighted regular relations, as probabilistic regular expressions.
  - Build up from atomic expressions a:b, with a in Σ^* and b in Δ^*
  - Concatenation, probabilistic union, probabilistic closure.
- Almost no work on this in NLP or ML, as far as I've seen.

Noisy Channel Model



Noisy Channel Model



Historical Note

- *Unweighted* FSTs were developed largely for designing and implementing models of the **morphology** of natural languages.
  - Huge amount of work at Xerox.
  - Also used in information extraction.
- Very useful for hand-construction of morphological rules individually, then assemble them by concatenation, union, composition, etc.

Parameterizations

1. Every arc gets one probability
2. Every “original” arc gets one probability
3. Log-linear distribution with shared features all over the WFST
   - This is really the most general, since features could be identities of arcs or of “original” arcs!

Exercise

- How to represent an HMM as a WFST?
  - MEMM?
  - Chain-structured CRF?
- How to represent stochastic edit distance as a WFST?
  - Elegant way to design a wide range of edit operations: composition of WFSTs

Back to Learning

- We want a *general* method learning the parameters from data, even when all layers are not known.
  - EM for HMMs is well known (Baum, 1972)
  - EM for stochastic edit distances is well known (Ristad and Yianilos, 1996)
- If our WFST was constructed by composing simpler machines, we might want to keep the original parameterization.
  - I.e., learn weights for individual machines jointly.

Very General Formulation of Learning

- Flexibility in parameterization, including
  - one probability per arc
  - one probability per "original" arc
  - log-linear distribution over arcs from a state, with parameter tying throughout the WFST
- Learn from samples of (input, output) pairs where $x_i \subseteq \Sigma^*$ and $y_i \subseteq \Delta^*$ (paths not observed).
  - supervised: each x_i and each y_i is a single string
  - unsupervised: $y_i = \Delta^*$
  - lots of possibilities in between

Maximum Likelihood Estimation

$$\max_{\boldsymbol{\theta}} \prod_i f_{\boldsymbol{\theta}}(x_i, y_i) \quad = \quad \max_{\boldsymbol{\theta}} \prod_i \sum_{\boldsymbol{a} \in Paths(x_i, y_i)} \prod_{j=1}^{|\boldsymbol{a}|} weight_{\boldsymbol{\theta}}(a_j)$$

- You can view this as a generalization of Baum-Welch training or EM for stochastic edit distances.
- Each example's total score is a **path sum** over the scores of paths that
  - are allowed by the WFST
  - match the input set x_i
  - match the output set y_i
- Recall that f might be a joint or conditional probability distribution and **θ** might be log-linear or multinomial parameters.

Expectation Maximization

- E step (one example): find the distribution over paths *given* x_i and y_i:

$$\frac{\sum_{\boldsymbol{a} \in Paths(x_i, y_i)} \prod_{j=1}^{|\boldsymbol{a}|} weight_{\boldsymbol{\theta}}(a_j)}{\sum_{\boldsymbol{a} \in Paths} \prod_{j=1}^{|\boldsymbol{a}|} weight_{\boldsymbol{\theta}}(a_j)}$$

- M step: update θ to make those paths more likely (exact form depends on parameterization).

$$\max_{\boldsymbol{\theta}} \sum_{a} \mathbb{E}_{\boldsymbol{\theta}^{(t-1)}} [freq(a)] \log weight_{\boldsymbol{\theta}}(a)$$

M Step

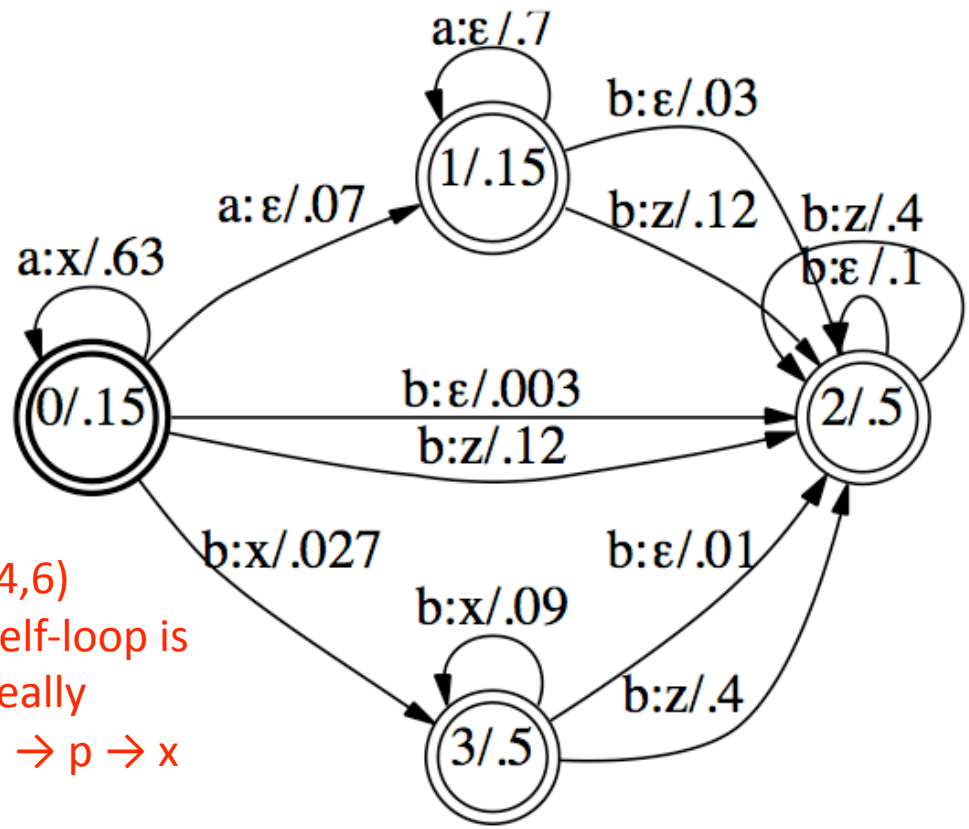
$$\max_{\boldsymbol{\theta}} \sum_{a} \mathbb{E}_{\boldsymbol{\theta}^{(t-1)}}[\mathit{freq}(a)] \log \mathit{weight}_{\boldsymbol{\theta}}(a)$$

- Parameterization 1 (one probability per arc):

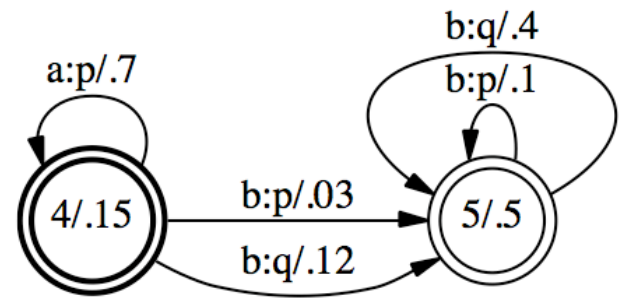
$$\max_{\boldsymbol{\theta}} \sum_{a} \mathbb{E}_{\boldsymbol{\theta}^{(t-1)}}[\mathit{freq}(a)] \log \theta_a$$

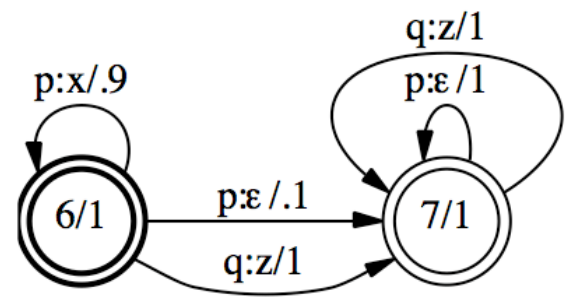
- Parameterization 2 (one probability per original arc): "unwind" p(a) into product of original arc probabilities.

“Unwinding” Example



(4,6)
self-loop is
really
a → p → x





0’s self-loop with a:x is really
4’s self-loop with a:p *and*
6’s self-loop with p:x

M Step

$$\max_{\boldsymbol{\theta}} \sum_{a} \mathbb{E}_{\boldsymbol{\theta}^{(t-1)}}[\mathit{freq}(a)] \log \mathit{weight}_{\boldsymbol{\theta}}(a)$$

- Parameterization 1 (one probability per arc):

$$\max_{\boldsymbol{\theta}} \sum_{a} \mathbb{E}_{\boldsymbol{\theta}^{(t-1)}}[\mathit{freq}(a)] \log \theta_a$$

- Parameterization 2 (one probability per original arc): "unwind" p(a) into product of original arc probabilities.
- Parameterization 3 (log-linear and most general):

$$\max_{\boldsymbol{\theta}} \sum_{a} \mathbb{E}_{\boldsymbol{\theta}^{(t-1)}}[\mathit{freq}(a)] \left(\boldsymbol{\theta}^{\top} \mathbf{g}(a) - \log \sum_{a' \in \mathit{Competitors}(a)} \exp \boldsymbol{\theta}^{\top} \mathbf{g}(a') \right)$$

Expectation Maximization

- E step (one example): find the distribution over paths *given* x_i and y_i:

$$\frac{\sum_{\boldsymbol{a} \in Paths(x_i, y_i)} \prod_{j=1}^{|\boldsymbol{a}|} weight_{\boldsymbol{\theta}}(a_j)}{\sum_{\boldsymbol{a} \in Paths} \prod_{j=1}^{|\boldsymbol{a}|} weight_{\boldsymbol{\theta}}(a_j)}$$

- M step: update θ to make those paths more likely (exact form depends on parameterization).

$$\max_{\boldsymbol{\theta}} \sum_{a} \mathbb{E}_{\boldsymbol{\theta}^{(t-1)}} [freq(a)] \log weight_{\boldsymbol{\theta}}(a)$$

E Step

$$\frac{\sum_{\boldsymbol{a} \in Paths(x_i, y_i)} \prod_{j=1}^{|\boldsymbol{a}|} weight_{\boldsymbol{\theta}}(a_j)}{\sum_{\boldsymbol{a} \in Paths} \prod_{j=1}^{|\boldsymbol{a}|} weight_{\boldsymbol{\theta}}(a_j)}$$

for a generative model, $Paths(\Sigma^*, \Delta^*)$
for a conditional model, $Paths(x_i, \Delta^*)$

- The likelihood value for one example is the ratio of two **path sums**.
  - The denominator path sum is the same for all examples in the generative case.
- But the E step's real job is to calculate sufficient statistics that the M step needs!

Flashback to October 4, when I talked about decoding …

Best Path

- General idea: take ***x*** and build a graph.
- Score of a path factors into the edges.

$$\arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) \quad = \quad \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \sum_{e \in \text{Edges}} \mathbf{f}(e) \mathbf{1}\{e \text{ is crossed by } \boldsymbol{y}\text{'s path}\}$$

- Decoding is finding the *best* path.

The Viterbi algorithm is an instance of finding a best path!

~~Best~~ Path Sum

- General idea: take ***x*** and build a graph.
- Score of a path factors into the edges.

$$\cancel{\arg\max_{\boldsymbol{y}}}\ \log \sum_{\boldsymbol{y}} \exp \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) = \cancel{\arg\max_{\boldsymbol{y}}}\ \log \sum_{\boldsymbol{y}} \exp \mathbf{w}^\top \sum_{e \in \text{Edges}} \mathbf{f}(e) \mathbf{1}\{e \text{ is crossed by } \boldsymbol{y}\text{'s path}\}$$

- ~~Decoding~~ scoring is finding the ~~*best*~~ path sum

The ~~Viterbi~~ forward algorithm is an instance of finding a ~~best~~ path sum!

Expected Feature Counts

- The E step's real job, in the most general case, is to calculate the expected feature counts in the examples, under the current model.
- We've seen this before!
  - Forward-backward; you can do it that way
  - Eisner suggests a different way, where the usual "plus-times" semiring is extended and the expectations are obtained in a single pass.

Semirings

The tuple $(K, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a **semiring** if:

- K is a set of values
- $\oplus$ is a commutative, binary operation $K \times K \rightarrow K$ with identity element **0**
- $\otimes$ is a binary operation $K \times K \rightarrow K$ with identity element **1**
  - for composition, $\otimes$ must be commutative
- $\otimes$ distributes over $\oplus$: $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$.
- **0** annihilates K: $a \otimes \mathbf{0} = \mathbf{0}$
- to handle infinite sets of cyclic paths, we need a unary closure operator * such that $a^* = \mathbf{1} \oplus a \oplus (a \otimes a) \oplus (a \otimes a \otimes a) \oplus \ldots$,

Semirings You Know

interpretation of weights	want to compute	weights	"plus"	"times"
probabilities	p(s)	[0, 1]	+	×
	best path prob.		max	×
log-probabilities	log p(s)	(-∞, 0]	log+	+
	best path log-prob.		max	+
costs	min-cost path	[0, +∞)	min	+
Boolean	s in language?	{0, 1}	∨	∧
strings	s itself	Σ*	set-union	concat.

The Expectation Semiring

- Instead of a weight storing just a "forward" score or probability, also store a value in V.
  - For us, V is vectors of (expected) feature counts.
  - Assign to each arc a value corresponding to its local feature vector.
- Define operations:

$$
\begin{aligned}
(p, \mathbf{v}) \otimes (p', \mathbf{v}') &= (pp', p\mathbf{v}' + p'\mathbf{v}) \\
(p, \mathbf{v}) \oplus (p', \mathbf{v}') &= (p + p', \mathbf{v}' + \mathbf{v}') \\
(p, \mathbf{v})^* &= (p^*, p^*\mathbf{v}p^*)
\end{aligned}
$$

- Result: final value contains path sum and feature expectations.

What's Really Happening?

- We are manipulating **weighted relations**, not WFSTs.
- The expectation semiring's values are scores and *gradients* of scores with respect to $\boldsymbol{\theta}$
  - Forward-backward (e.g., for CRFs) is doing the same thing, only using the chain rule for derivatives to define a second pass (the backward pass).
  - The expectation semiring lets you avoid the backward pass and the per-arc products of forward and backward probabilities.
  - But it's probably slower in practice.

Other Goodies in the Paper and Later Work

- Analysis as the "algebraic path" problem, links to a range of speedups (e.g., for acyclic graphs).
- Viterbi variant of the expectation semiring.
- Probabilistic regular expressions idea.
  - Potential for rapid incorporation of expert intuitions into data-driven systems?
- Li and Eisner (2009) goes to *second-order* expectation semirings!
- Dreyer and Eisner (2009) uses WFSTs to define factors in graphical models!

Closing Notes on Learning

- Eisner's approach is for MLE (and MAP), but his algorithms are actually inference methods for WFSTs.
  - By changing to maximization and incorporating costs, you can do perceptron, structured SVM, and other error-driven learning.
- We didn't talk at all about learning the structure of finite-state models!
  - There's a rich formal literature on this, and not too many papers that attempt it for real problems.
  - I gave one classic citation on the wiki (Stolcke and Omohundro, 1993).

Toolkits

- FSM libraries (AT&T)
  - Free binaries
  - Implements pretty much everything you need to build weighted and unweighted FS recognizers and transducers … except training!
- Xerox FS toolkit
  - Web demo; software can be purchased
  - No weights
- RWTH FSA toolkit
  - Newer, open-source
  - Not sure what's implemented
- OpenFST (Google)
  - New incarnation of FSM libraries
  - Free and open source!

Notes on Algorithms

	limitations	polytime
ε-removal Mohri 2002		✓
determinize Mohri 1997	not all transducers	
minimize Eisner 2003	not all semirings	✓

Conclusion

- WFSTs are extremely general and powerful
  - People use them to implement or approximate almost everything in NLP, IE, and MT
- You should know this abstraction, even if you don't use it every day.