Engineering Part IIB: Module 4F11
Speech and Language Processing
Lectures 9 & 10: Weighted Finite State Transducers for Speech and Language Processing

Bill Byrne

Lent 2016

Cambridge University Engineering Department

Introduction

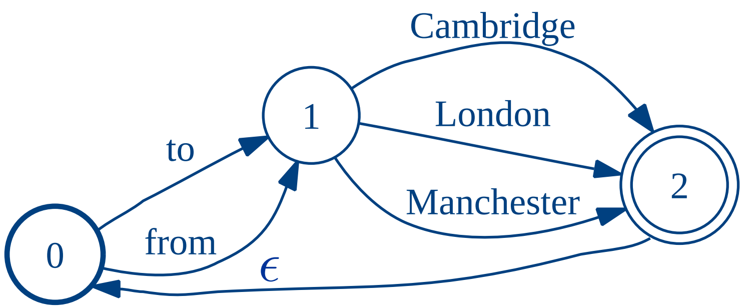
Weighted finite state machines are machines which accept strings of symbols. They are limited in their power, e.g. they can accept regular expressions such as $a^n b^m$ but not $a^n b^n$. Despite their limitations, they can be very powerful tools for speech and language processing. In particular, they are very well-suited for carrying out search procedures involving Markov processes and hidden Markov models. If it is possible to cast a problem in a WFSA framework, standard algorithms can be applied directly to the problem.

Topic outline for Lectures 9 and 10

1. Examples of Finite State Automata
2. Weighted Finite State Acceptors
3. Example 1: hypothesis testing
4. Example 2: WFSAs and N-Gram LMs
5. Operations on WFSAs
6. Weighted Finite State Transducers
7. WFSTs as ASR components
8. Operations on WFSTs

Uses for Finite State Automata - Simple Grammars

Unweighted acceptors can be used to define simple grammars. In these grammars
- no differentiation between strings which are accepted
- strings which are not accepted are not recognized in the application



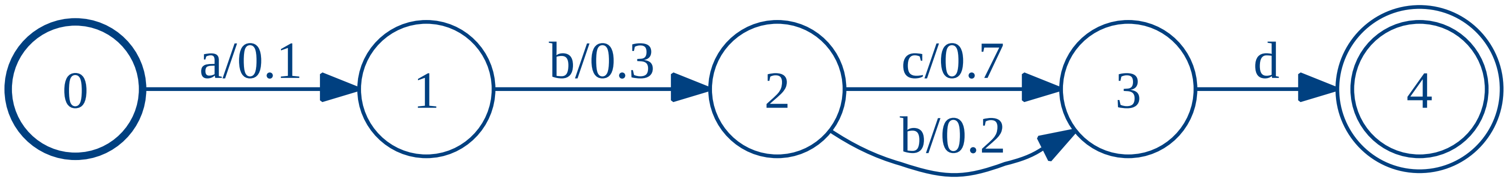
In this example
- ‘to Cambridge from London’ and ‘from London to Cambridge’ are accepted
- ‘to Paris’ is rejected
- ‘to Cambridge to London’ is also accepted

Natural for many simple speech recognition applications, e.g. digit dialing

Uses for Finite State Automata - Weighted Acceptors

Weighted acceptors can assign costs to strings
- weights (or costs) are accumulated over **paths** through the automata
- a path is a sequence of edges (or arcs)
- strings are associated with paths

A weighted automaton which accepts only the strings 'a b c d' and 'a b b d' :

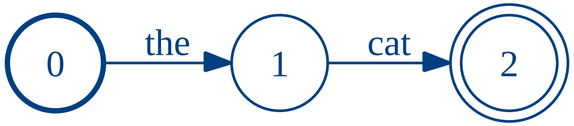


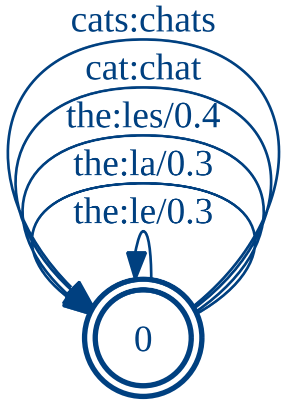
$$w(\text{'a b c d'}) = 0.1 + 0.3 + 0.7 + 0.0 = 1.1$$
$$w(\text{'a b b d'}) = 0.1 + 0.3 + 0.2 + 0.0 = 0.6$$

To define the acceptor, we specify a set of states Q and a set of arcs : $q \stackrel{x/k}{\rightarrow} q'$
- q is the start state, q' is the end state, x is the input symbol, k is the arc weight
- e.g. for the second arc, $q = 1$, $q' = 2$, $x = b$, $k = 0.3$: $1 \stackrel{b/0.3}{\rightarrow} 2$

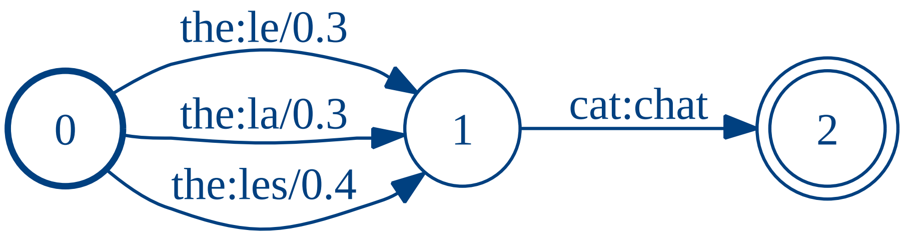
Uses for Finite State Automata - Weighted Transducers

An acceptor and a weighted transducer:





Their **composition** (more on this later):



the cat : le chat / 0.3
the cat : la chat / 0.3
the cat : les chat / 0.4

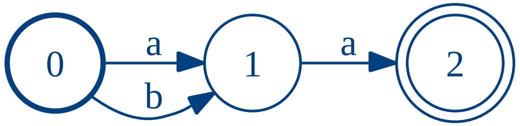
To describe the transducer, we specify a set of states and a set of arcs : $q \stackrel{x:y/k}{\rightarrow} q'$
- x is the input symbol, y is the output symbol, k is the arc weight

Assigning weights to paths is useful to describe ambiguity and uncertainty

Strings and Automata

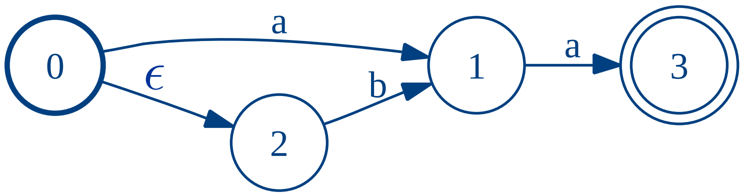
Suppose we have an alphabet Σ . Σ^* is the set of sequences drawn from Σ .
- e.g. $\Sigma = \{a, b\}$. $\Sigma^* = \{$ 'a', 'b', 'aa', 'ab', 'ba', 'bb', 'aaa', 'aab', ... $\}$

An automata can be thought of as either an *acceptor* or a *generator*



- acceptor: only 'a a' or 'b a' lead to the final state
- generator: valid paths generate either 'a a' or 'b a'

'epsilon' arcs allow transitions which do not consume any input symbols



- this machine is equivalent to the one above; it accepts either 'a a' or 'b a'
- useful for 'glueing' automata together

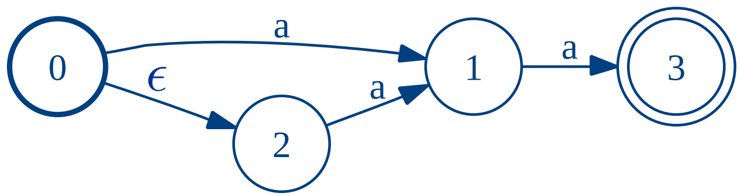
- because of epsilons, paths which accept a sequence may differ in length

Strings and Automata [2]

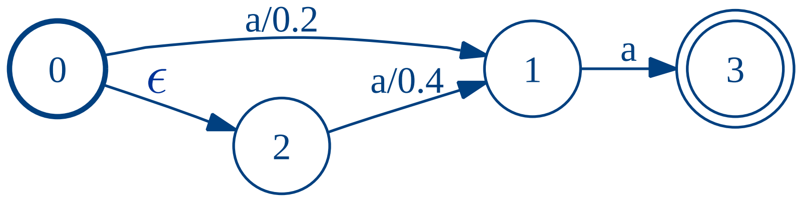
Sequences can be 'accepted' by more than one path through the acceptor

For unweighted acceptors this does not pose any problems.

- since the only question is whether or not an unweighted transducer accepts a sequence, the presence of alternative paths does not introduce any ambiguity
- e.g. the following is a valid acceptor: there are two paths which accept 'a a'



The following weighted acceptor is also valid



- there are two paths, one with weight 0.2 and the second with weight 0.4

Problem: What weight should be assigned to 'a a' ?

Solution: 'Sum' the weights of all paths which accept 'a a' .

Weighted Finite State Acceptors - Definition

A weighted acceptor over a finite input alphabet Σ is a finite directed graph with a set of nodes Q (states) and a set of arcs E (edges).

- Each arc (or edge) e has an initial (or start) state $s(e)$ and a final state $f(e)$.
- Each arc e is labeled with an input symbol $i(e)$ and a weight $w(e)$.
- The weights take values in $\mathbb{K}$

A complete **path** through an acceptor can be written as $p = e_1 \cdots e_{n_p}$, where :

- the path p consists of n_p edges
- the path starts at state $i_p = s(e_1)$ where i_p is an initial state
- the path ends at state $f_p = f(e_{n_p})$ where f_p is a final state

The arc weights and initial and final weights combine to form the **path weight**

$$w(p) = \lambda(i_p) \ \otimes \ w(e_1) \ \otimes \cdots \otimes \ w(e_{n_p}) \ \otimes \rho(f_p)$$

- Initial weights and final weights : $\lambda(i_p)$ and $\rho(f_p)$
- $\otimes$ is the **product** of two weights (to be defined shortly)

Notation: $\otimes_{j=1}^{np} w(e_j) = w(e_1) \otimes \cdots \otimes w(e_{n_p})$ so that $w(p) = \lambda(i_p) \otimes (\otimes_{j=1}^{np} w(e_j)) \otimes \rho(f_p)$

Weights Assigned to Strings by Acceptors

Since each arc in the WFSA has an input symbol, it is straightforward to associate paths through the acceptor with input sequences.
- A path $p = e_1 \cdots e_{n_p}$ produces the string $x = i(e_1) \cdots i(e_{n_p})$

If every string was generated by a unique path through an acceptor, assigning weights to strings would be easy: the weight of a string would be its path weight. However, since strings can be generated by multiple paths, the acceptor combines the weights of all paths which might have generated a string, as follows:

- Let x be a string constructed from symbols in the input alphabet Σ : $x \in \Sigma^*$
- Let $P(x)$ be the set of complete paths which generate x, i.e. $x = i(e_1) \cdots i(e_{n_p})$
- Let $\oplus$ be the *sum* of two weight values
- Define $[\![A]\!](x)$ as the cost assigned to the string x by the transducer

$$[\![A]\!](x) = \bigoplus_{p \in P(x)} \underbrace{\lambda(i_p) \otimes (\otimes_{j=1}^{n_p} w(e_j)) \otimes \rho(f_p)}_{w(p)}$$

$[\![A]\!](x)$ is the 'Sum' of the weights of the complete paths which can generate x

Weights and Operations on Weights

The *product* operation $\otimes$ is used to compute the weight of a single path from the weights of its edges

The *sum* operation $\oplus$ is used to compute the weight of a sequence by summing over all the distinct paths which could have generated that sequence

Semirings : sum $\oplus$ and product $\otimes$ with identity elements $\bar{0}$ and $\bar{1}$
- For a weight $k \in \mathbb{K}$: $\bar{0} \oplus k = k$; $\bar{1} \otimes k = k$; $\bar{0} \otimes k = \bar{0}$
- $\oplus$ and $\otimes$ distribute and commute in the familiar way

Three useful semirings:

Semiring	$\mathbb{K}$	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Probability	$\mathbb{R}_+$	$+$	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, \infty\}$	$\oplus_{\log}$	$+$	∞	0
Tropical	$\mathbb{R} \cup \{-\infty, \infty\}$	$\min$	$+$	∞	0

$\oplus_{\log} : k_1 \oplus_{\log} k_2 = -\log(e^{-k_1} + e^{-k_2})$

Unless otherwise stated, the tropical semiring is used by default

Example 1: Generating ‘a b’ Under Two Hypotheses

Suppose we have two probability distributions over the string ‘a b’

$$\text{Hypothesis 1}: \; P(\text{‘a b’}, h_1) = p_1(\text{‘a b’})\, p_1$$
$$\text{Hypothesis 2}: \; P(\text{‘a b’}, h_2) = p_2(\text{‘a b’})\, p_2$$

We may be interested in the **marginal probability** :

$$P(\text{‘a b’}) = p_1(\text{‘a b’})\, p_1 + p_2(\text{‘a b’})\, p_2$$

- probability of generating ‘a b’ under either hypothesis

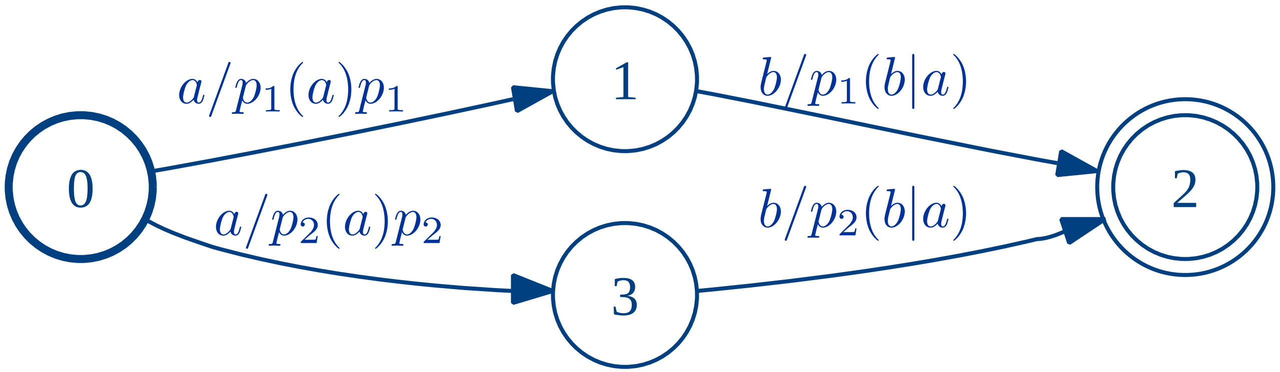
Alternatively, we may be interested in the hypothesis which assigns the highest likelihood to the sequence, sometimes called the **Viterbi score** :

$$\max_{i=1,2} p_i(\text{‘a b’})\, p_i$$

By setting the weights appropriately and choosing the right semiring, these quantities can be computed by WFST operations

Example 1: Weights Under the Probability Semiring

Find the weight assigned to the string ‘a b’ by the following acceptor:

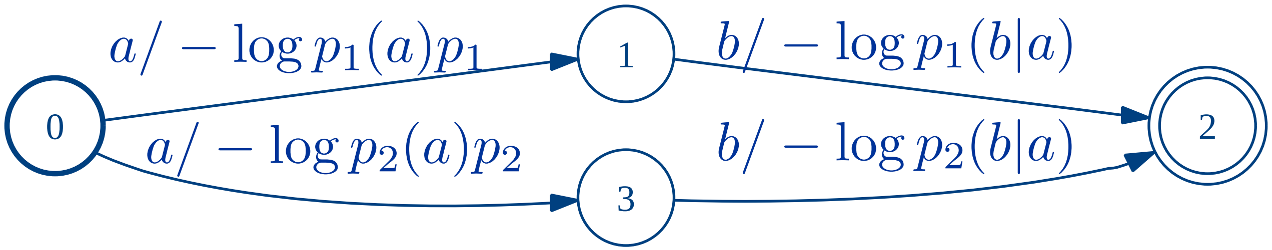


- Operations on weights via ‘usual’ multiplication and addition: $(\oplus, \otimes) = (+, \times)$

$$
\begin{aligned}
[\![A]\!](\text{‘a b’}) &= \underbrace{w(e_1) \otimes w(e_2)}_{\text{top path}} \oplus \underbrace{w(e_3) \otimes w(e_4)}_{\text{bottom path}} \\
&= p_1(a)p_1 \times p_1(b|a) + p_2(a)p_2 \times p_2(b|a) \\
&= p_1(\text{‘a b’})\, p_1 + p_2(\text{‘a b’})\, p_2 \;\Leftarrow\; \text{marginal probability}
\end{aligned}
$$

Example 1: Weights Under the Log Semiring

Find the weight assigned to the string 'a b' by the following acceptor:
- weights are negative log likelihoods

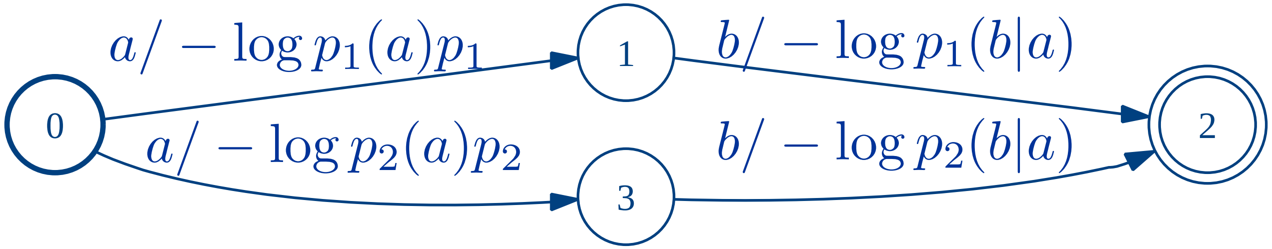


- Weight operations: $(\oplus, \otimes) = (\oplus_{\log}, +)$ where $\oplus_{\log} : k_1 \oplus_{\log} k_2 = -\log(e^{-k_1} + e^{-k_2})$

$$
\begin{aligned}
[\![A]\!](\text{'a b'}) &= w(e_1) \otimes w(e_2) \oplus w(e_3) \otimes w(e_4) \\
&= [-\log p_1(a)p_1 - \log p_1(b|a)] \oplus_{\log} [-\log p_2(a)p_2 - \log p_2(b|a)] \\
&= [-\log p_1(\text{'a b'})p_1] \oplus_{\log} [-\log p_2(\text{'a b'})p_2] \\
&= -\log[\exp(\log p_1(\text{'a b'})p_1) + \exp(\log p_2(\text{'a b'})p_2)] \\
&= -\log[\, p_1(\text{'a b'})\, p_1 + p_2(\text{'a b'})p_2\,] \;\Leftarrow\; \text{negative log marginal prob}
\end{aligned}
$$

Example 1: Weights Under the Tropical Semiring

Find the weight assigned to the string ‘a b’ by the following acceptor:



- Weight operations: $(\oplus, \otimes) = (\min, +)$ where $\min : k_1 \min k_2 = \min(k_1, k_2)$

$$
\begin{aligned}
[\![A]\!](\text{‘a b’}) &= w(e_1) \otimes w(e_2) \;\oplus\; w(e_3) \otimes w(e_4) \\
&= \min[-\log p_1(\text{‘a b’})\, p_1\,,\; -\log p_2(\text{‘a b’})\, p_2] \\
&= \underbrace{-\max\,[\log\; p_1(\text{‘a b’})\, p_1\,,\; \log\; p_2(\text{‘a b’})\, p_2]}_{\text{negative log Viterbi likelihood}}
\end{aligned}
$$

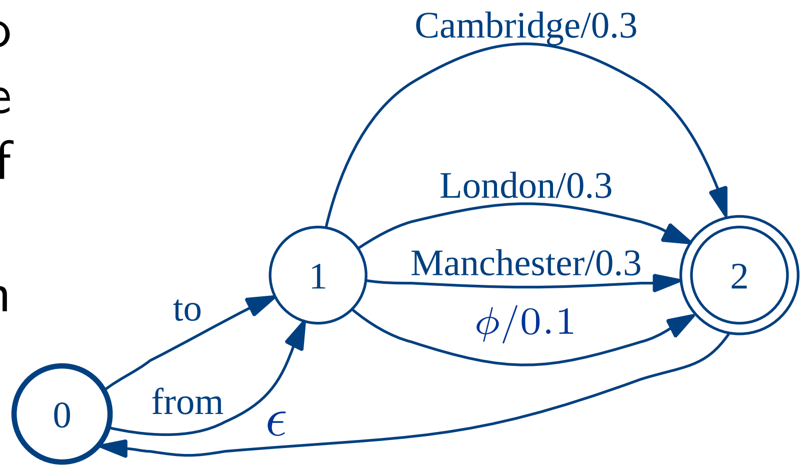
Example 2: WFSAs and N-Gram Language Models

WFSAs can be used to implement N-Gram language models, but the 'back-off' is problematic. Recall the back-off bigram language model:

$$\hat{P}(w_j|w_i) = \begin{cases} p(w_i, w_j) & f(w_i, w_j) > C \\ \alpha(w_i)\hat{P}(w_j) & \text{otherwise} \end{cases}$$

where $p(w_i, w_j) = d(f(w_i, w_j))\frac{f(w_i,w_j)}{f(w_i)}$. As described so far, WFSAs do not have the ability to implement an 'otherwise' and therefore it is difficult to implement a back-off n-gram directly.

'Failure transitions' are introduced to deal with such problems. A failure transition is labelled by ϕ and is taken if and only if no other arc can be taken.
- e.g. at right, 'to paris' is accepted with weight w('to paris') $= 0.1$.



Example 2: WFSAs and N-Gram Language Models

Back-off N-Gram language models can be encoded using failure transitions. The following describes a bigram implementation.

- There is one state for every word, plus a unigram back-off state ϵ :

$$Q = \{(w_1), \ldots, (w_V), \epsilon\}$$

- There is an arc for each pair of words w and w' for which $f(w, w') > C$

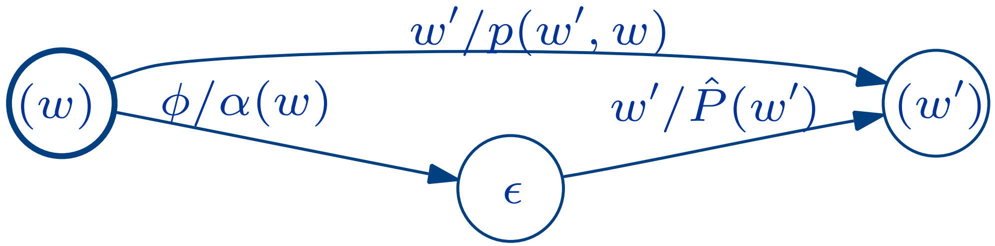
$$(w) \overset{w' / \, p(w'|w)}{\longrightarrow} (w')$$

- There is a back-off arc from every word state (w) to the backoff state ϵ

$$(w) \overset{\phi / \, \alpha(w)}{\longrightarrow} \epsilon$$

- There is a unigram arc from the back-off state ϵ to every word state (w')

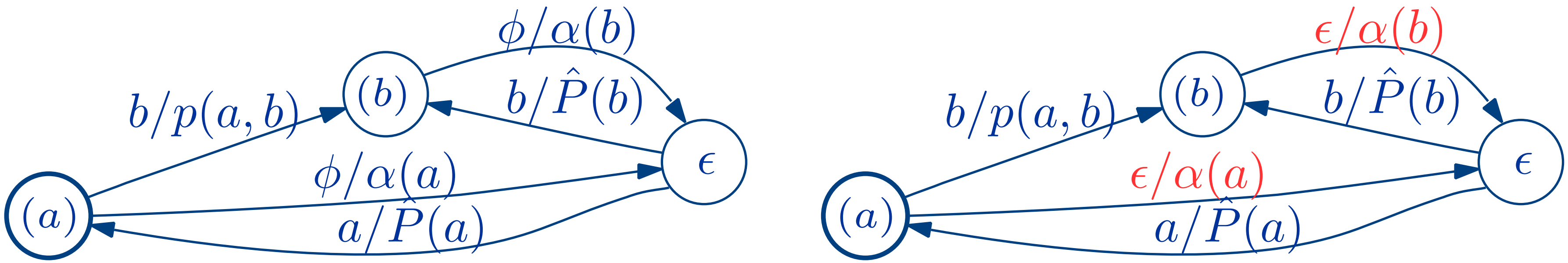
$$\epsilon \overset{w' / \hat{P}(w')}{\longrightarrow} (w')$$



Example 2: A Small Back-off Bigram Language Model

Language model vocabulary : $\Sigma = \{a, b\}$. WFST states : $Q = \{(a), (b), \epsilon\}$
Cutoff statistics : $f(a, b) > C$ but $f(b, a) < C$
Bigram probabilities:

$P(b|a) = p(a, b)$ $\leftarrow$ discounting, but no back-off
$P(a|b) = \alpha(b)\hat{P}(a)$ $\leftarrow$ back-off



Exact Implementation Approximate Implementation

Implementing the failure transition can be complicated, so an approximate implementation is sometimes used which substitutes epsilons for the failure arcs. The flaw is that the back-off path can always be taken, even when a non-back-off path is present.

WFSA Operations

Basic operations can be performed over WFSAs

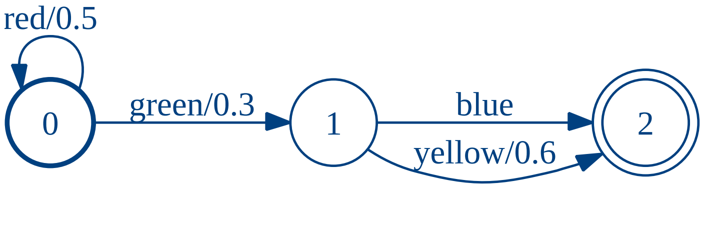
Some operations correspond to operations on the languages defined by WFSAs :
- Intersection
- Union
- Concatenation (or Product)
- ...

Other operations correspond to operations on the WFSA itself :
- Determinization
- Shortest distance calculations
- ...

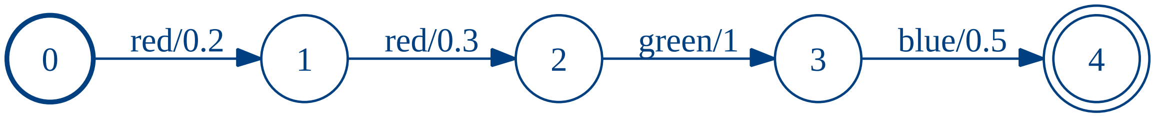
WFSA Operations - Intersection

A string x is accepted by $C = A \cap B$ if x is accepted by A and by B

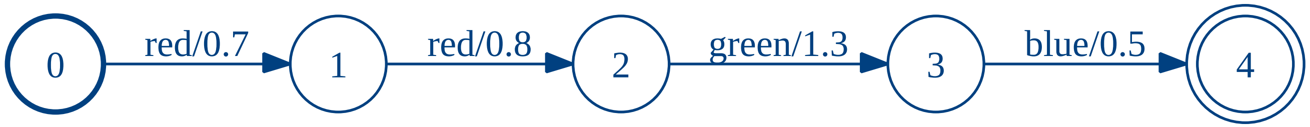
$$[\![C]\!](x) = [\![A]\!](x) \otimes [\![B]\!](x)$$



A



B



$C = A \cap B$

In this example $x =$ 'red red green blue' and $(\oplus, \otimes) = (\min, +)$.

Verify that $[\![A \cap B]\!](x) = [\![C]\!](x)$:

$[\![A]\!](x) = 0.5 + 0.5 + 0.3 + 0.0 = 1.3$

$[\![B]\!](x) = 0.2 + 0.3 + 1 + 0.5 = 2.0$

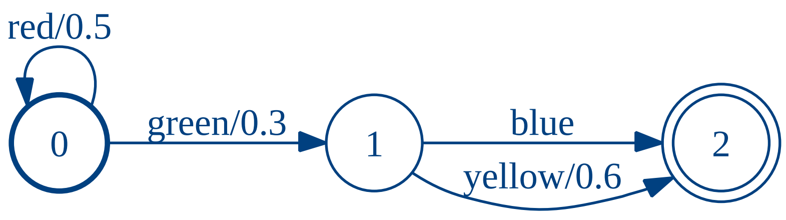
$[\![C]\!](x) = 0.7 + 0.8 + 1.3 + 0.5 = 3.3$

$[\![A \cap B]\!](x) = [\![A]\!](x) \otimes [\![B]\!](x) = [\![A]\!](x) + [\![B]\!](x) = 1.3 + 2.0 = 3.3$

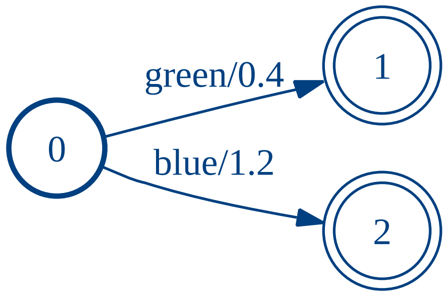
WFSA Operations - Union

A string x is accepted by $C = A \cup B$ if x is accepted by A or by B

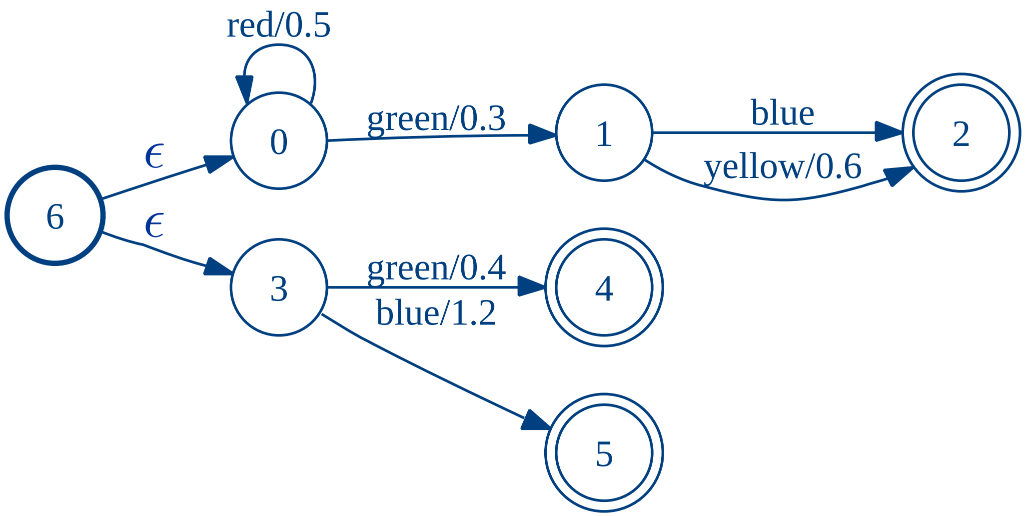
$$[\![C]\!](x) = [\![A]\!](x) \oplus [\![B]\!](x)$$



A



B

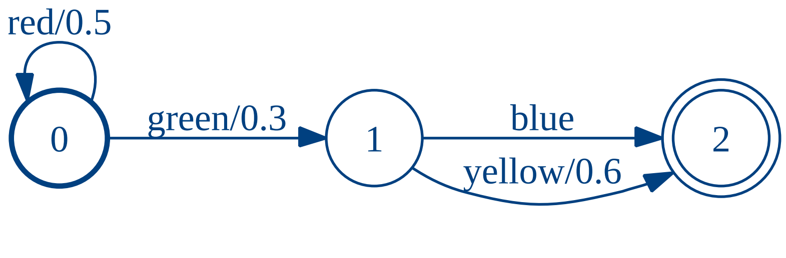


$C = A \cup B$

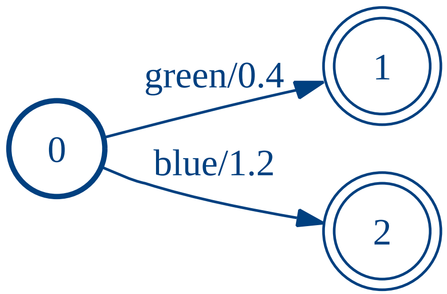
WFSA Operations - Concatenation (or Product)

A string x is accepted by $C = A \otimes B$ if x can be split into $x = x_1 x_2$ such that x_1 is accepted by A and x is accepted by B

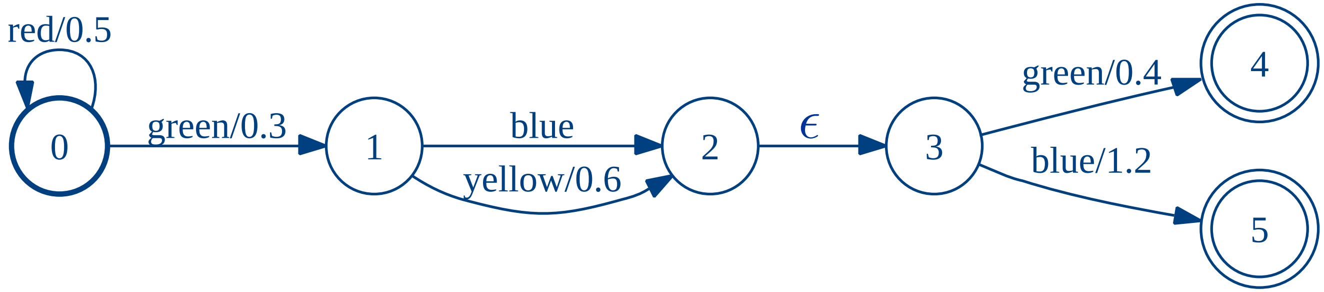
$$[\![C]\!](x) = \bigoplus_{x_1, x_2 :\, x = x_1 x_2} [\![A]\!](x_1) \otimes [\![B]\!](x_2)$$



A



B

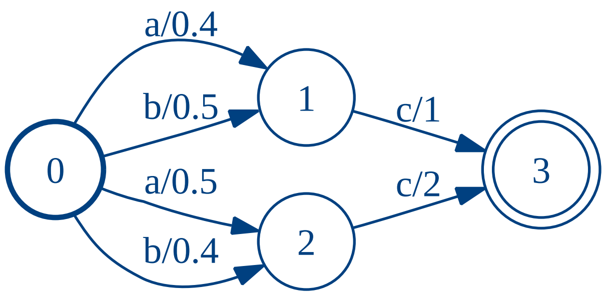


$C = A \otimes B$

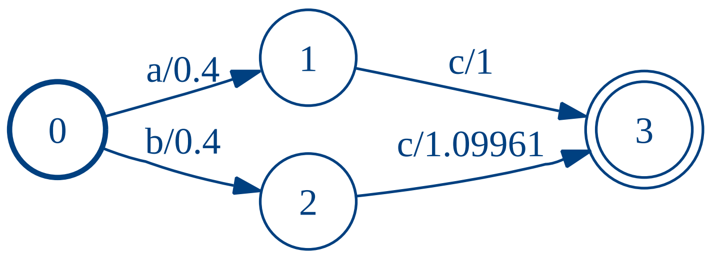
WFSA Operations - Determinization

Some WFSAs (in some semirings) can be **determinized**. After determinization:

- there is a unique starting state
- no two transitions leaving a state share the same input label
- arc weights may change, but weights assigned to strings are unchanged
- there may be many new epsilon arcs



Before Determinization



After Determinization

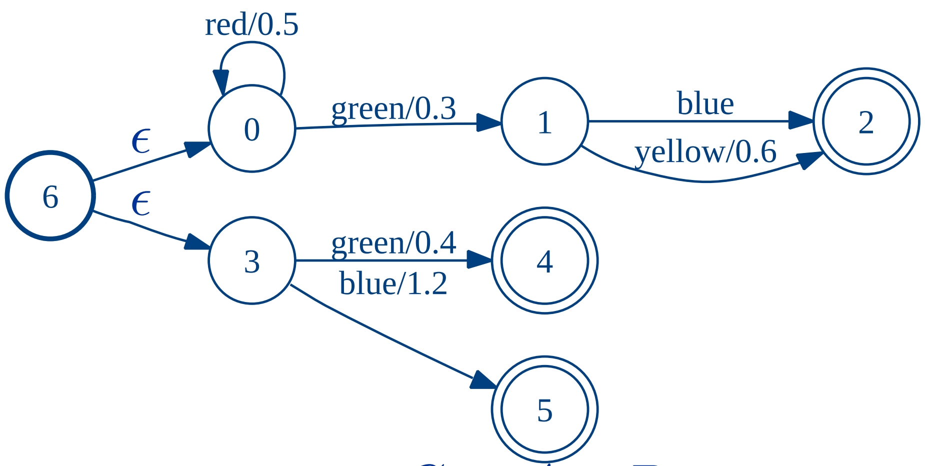
- determinization can be followed by **minimization** which finds an equivalent machine with a minimal number of states and arcs

WFSA Operations - Single Shortest Distance Algorithms

Let F be the set of final states (in case there's more than one)
Let $P(q, F)$ be the set of paths from any state q to any final state in F
- $d[q]$ is the sum of the weights of all paths from q to any final state in F

$$d[q] = \bigoplus_{p \in P(q,F)} w(p)$$

- the costs $d[q]$ can be computed efficiently (e.g. recursively), and trace-back can be added to reconstruct shortest-distance paths



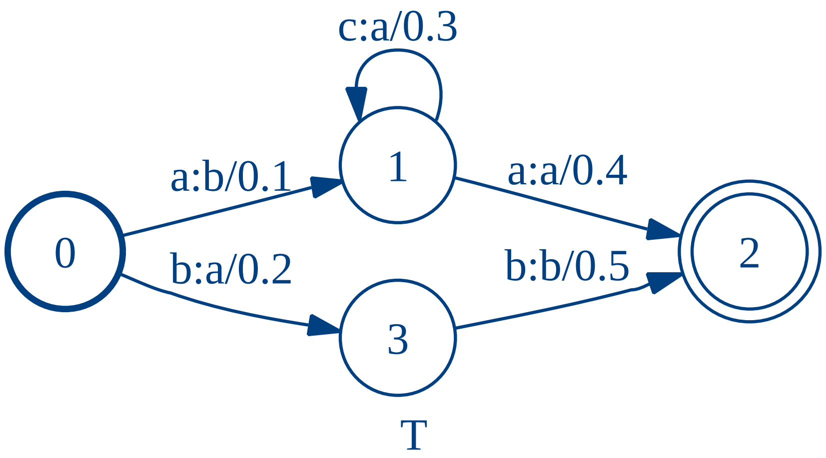
$C = A \cup B$

```
> fstshortestdistance --reverse C.fst
0 0.30
1 0
2 0
3 0.40
4 0
5 0
6 0.30
```

Leads easily to a **least cost calculation** procedure
- e.g. the weight of the shortest complete path is 0.30 .

Weighted Finite State Transducers

WFSTs can be used to transform one string to another string
- this is done via symbol-to-symbol mappings
- arcs are modified to have an 'output' symbol
- the interpretation is 'read a symbol x , write a symbol y'
- weights are applied analogously to weighted acceptors



Input String x	Output String y	Cost $[\![T]\!](x, y)$
'b b'	'a b'	0.7
'a a'	'b a'	0.5
'a c a'	'b a a'	0.8
'a c c a'	'b a a a'	1.1
'a c c c a'	'b a a a a'	1.4
$\vdots$	$\vdots$	

In a weighted transducer, arcs have the form: $q \overset{x:y/k}{\rightarrow} q'$
- e.g. the WFST T has an arc with $q = 0$, $q' = 3$, $x = b$, $y = a$, $k = 0.2$

Weighted Finite State Transducer – Definition

The definition of the acceptor is extended to support output operations:
- Two alphabets: Input alphabet: Σ , Output alphabet: Δ
- Each arc (edge) e has an output symbol $o(e) \in \Delta$
- Each arc e has an input symbol $i(e) \in \Sigma$
- For strings $x \in \Sigma^*$ and $y \in \Delta^*$, define $P(x, y)$ to be the set of all complete paths $p = e_1 \cdots e_{n_p}$ which have x as an input sequence and y as an output sequence

$$p \in P(x, y) : x = i(e_1) \cdots i(e_{n_p}) \, , \; y = o(e_1) \cdots o(e_{n_p})$$

- Path weights are computed as in acceptors: $w(p) = \otimes_{j=1}^{n_p} w(e_j)$

The transducer T implements a **weighted mapping** of string x to string y :
- the weight is the sum of all path weights along which x is mapped to y

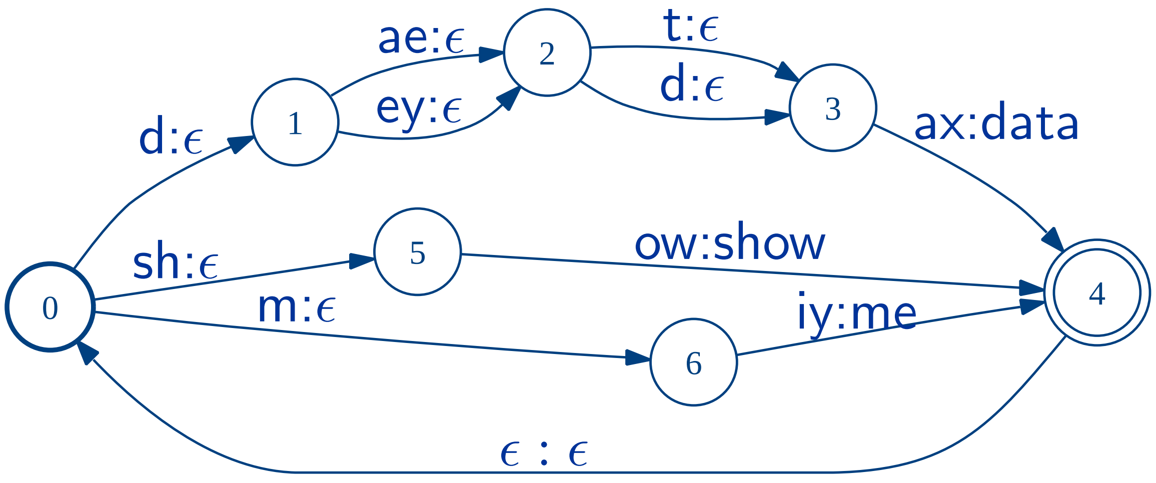
$$[\![T]\!](x, y) = \bigoplus_{p \in P(x,y)} w(p)$$

Example 3: WFST Pronunciation Lexicon

Suppose we have a pronunciation lexicon with the following entries:

Word	Pronunciation
data	d ey t ax d ey d ax d ae t ax d ae d ax
show	sh ow
me	m iy

The following transducer maps **phone sequences** to **word sequences**



L

Example 4: WFST Context-Dependent Triphone Transducers

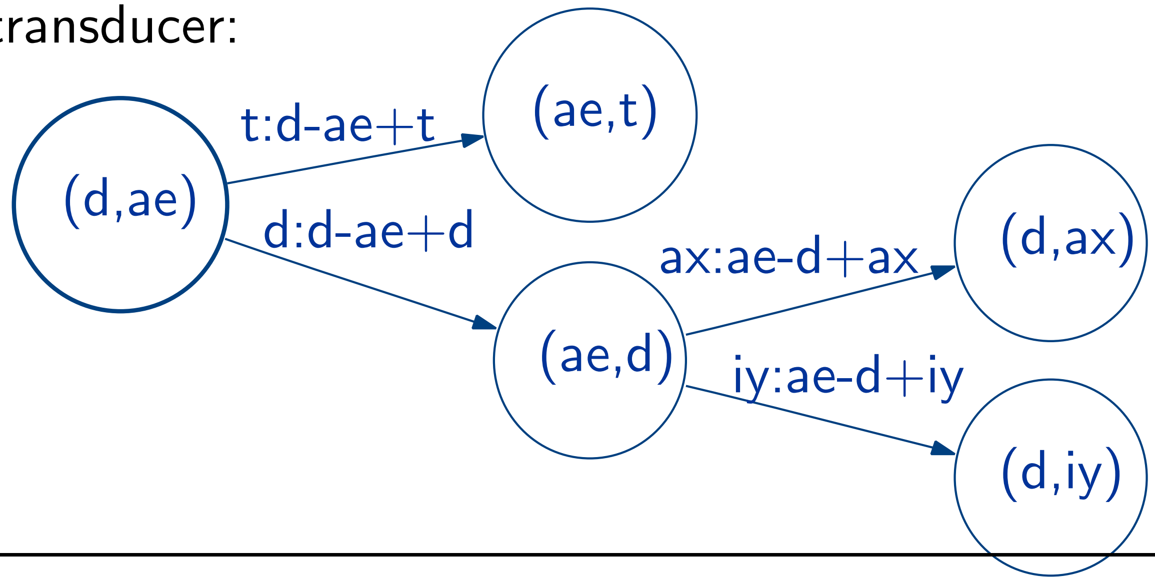
A CI-to-CD transducer maps **monophone sequences** to **triphone sequences**.
- e.g. the transducer should map '... d ae t ax ...' to '... d-ae+t ae-t+ax ...'

States are added to keep track of the phonetic context, as follows:
For every three monophones, p_1, p_2, p_3 :
- add the states (p_1, p_2) and (p_2, p_3) to the transducer
- for the triphone $t = p_1$-p_2+p_3 , add the following arc between the two states:

$$(p_1, p_2) \overset{p_3:t}{\rightarrow} (p_2, p_3)$$

- Silence models, monophones, etc must be handled differently

Fragment from a CI-to-CD transducer:

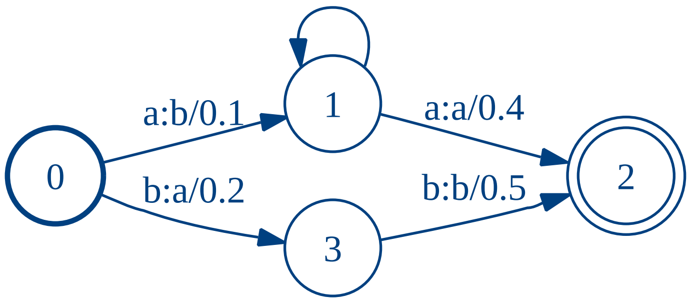


WFST Operations – Composition

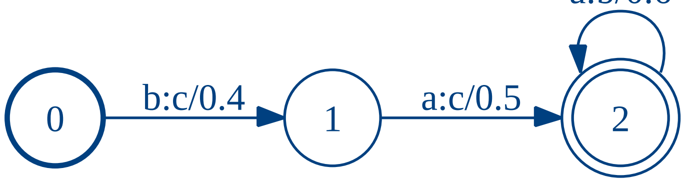
Suppose A and B are two WFSTs: A maps x to y ; B maps y to z.

$A \circ B$ is the composition of A with B which maps x to z

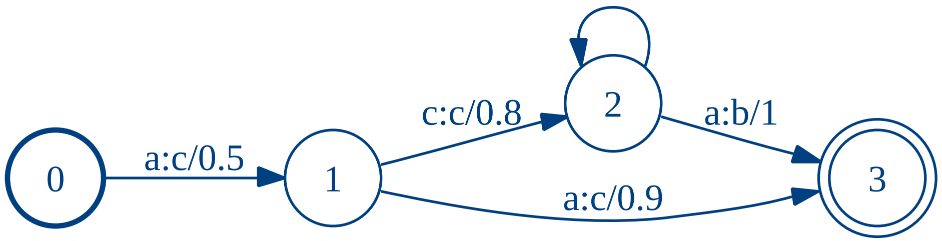
$$[\![A \circ B]\!](x,z) = \bigoplus_y [\![A]\!](x,y) \otimes [\![B]\!](y,z)$$



A



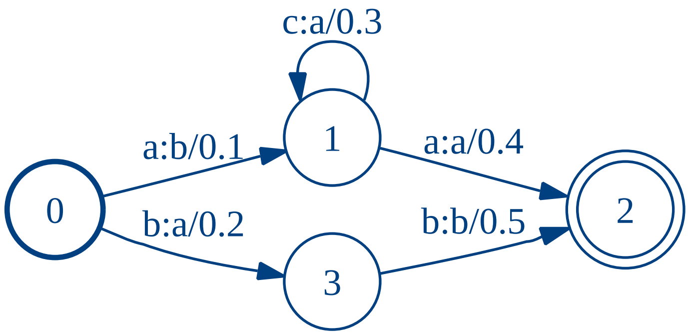
B



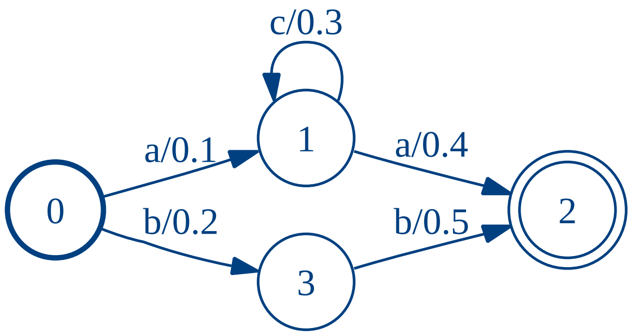
$A \circ B$

WFST Operations – Projection

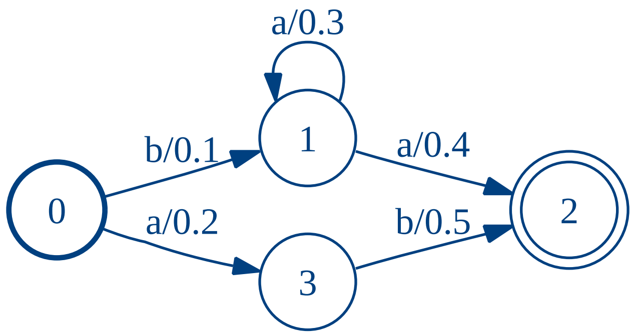
Transforms a transducer to an acceptor by projecting either onto the input arcs or the output arcs.



A



A_1

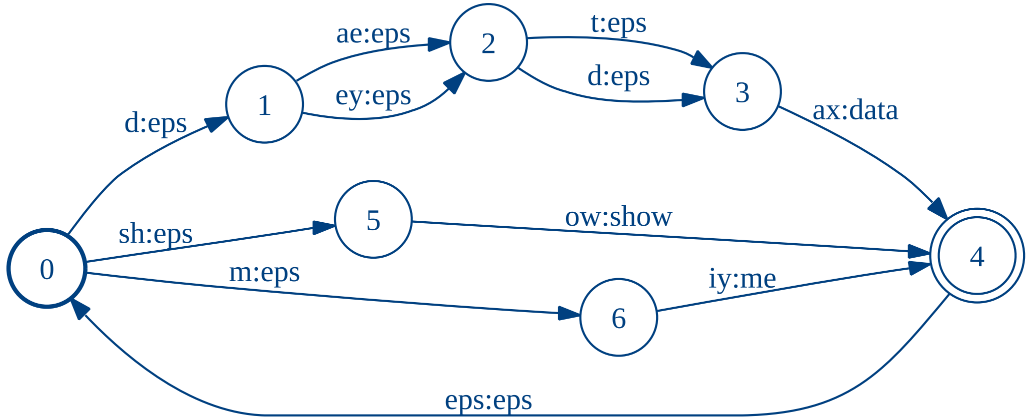


A_2

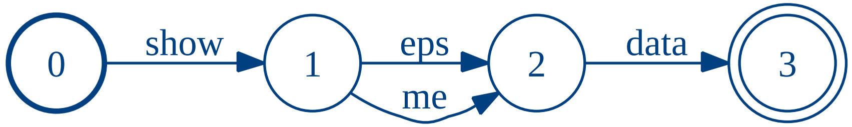
Create A_1 by **input projection** of A : $[\![A_1]\!](x) = \bigoplus_y [\![A]\!](x,y)$

Create A_2 by **output projection** of A : $[\![A_2]\!](y) = \bigoplus_x [\![A]\!](x,y)$

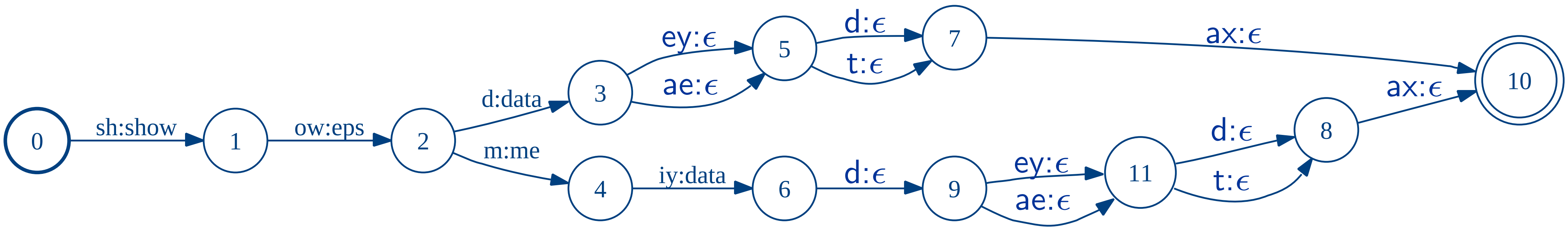
Example 5: Monophone Network for a Constrained Task



L



Word Recognition Task



Monophone Recognition Network

Monophone recognition network created by composition of the pronunciation transducer and the word recognition task acceptor.

Example 6: ASR Recognition Networks

Suppose we wish to build an ASR system based on a set of acoustic triphone HMMs, a pronunciation lexicon, and an n-gram language model. A 'recognition network' can be constructed by composing the transducers for these entities.

Notation: M - monophone sequences , and T - triphone sequences

$$
\begin{aligned}
\operatorname*{argmax}_W P(O,W) &= \operatorname*{argmax}_W \sum_{T,M} P(O|T,M,W)P(T|M,W)P(M|W)P(W) \\
&\approx \operatorname*{argmax}_W \max_{T,M} P(O|T)\, P(T|M)\, P(M|W)\, P(W) \\
&= \operatorname*{argmax}_W \max_T P(O|T)\, \max_M P(T|M)\, P(M|W)\, P(W) \\
&= \operatorname*{argmax}_W \max_T - P(O|T)\, [\![N]\!](T,W)
\end{aligned}
$$

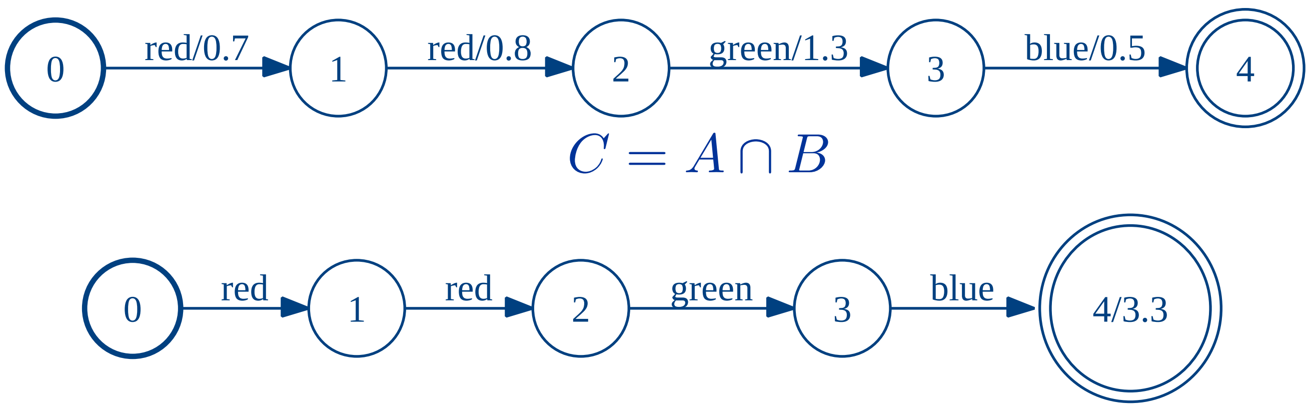
The transducer N maps triphone sequences T to word sequences W. N is the composition of the CD-to-CI transducer, the pronunciation lexicon transducer, and the language model acceptor, with composition under the tropical semiring.

Pushing

Arc weights and labels can be moved if weights assigned to strings are preserved.

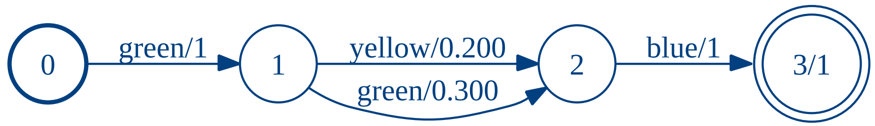
Pushing moves weights and/or labels towards the start or the end state
- pushing towards the start state can improve pruning
- pushing towards the end states can help accumulating costs over paths

Pushing weights towards final states:
- for each state, the sum of the weights of incoming arcs must equal $\bar{1}$
- recall that final states can also have weights

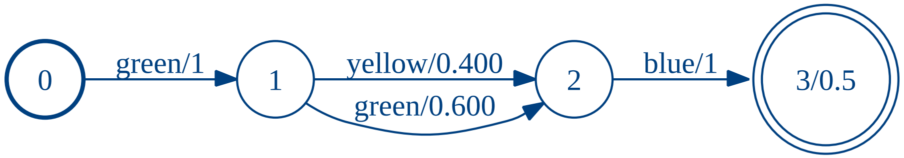


Weight pushing -- Tropical Semiring

Pushing (2)



Weights are probabilities



Weight Pushing -- Real Semiring

- Final state has sum of all path weights
- Arc weights have posterior probabilities

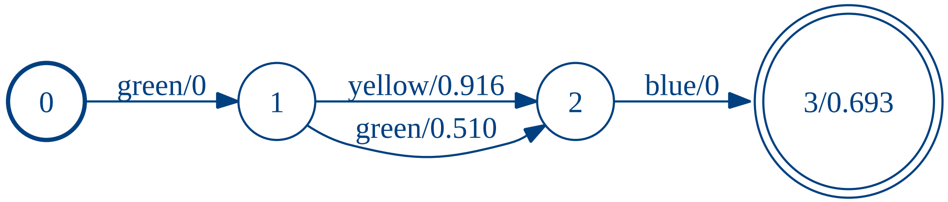
Q: What is the posterior probability that any path contains 'yellow' ?

A: 0.4

Weights are negative log probabilities

$\log 0.3 = -1.203 \quad \log 0.2 = -1.609$

Negative Log Weights



Weight Pushing -- Log Semiring

$\log 0.6 = -0.511 \quad \log 0.4 = -0.9163 \quad \log 0.5 = -0.69314$

- Costs are negative log probabilities

Recursive Transition Networks

An RTN is a **family** of FSAs. Formally, $R = (\mathbb{N}, \Sigma, (T_\nu)_{\nu \in \mathbb{N}}, S)$, where

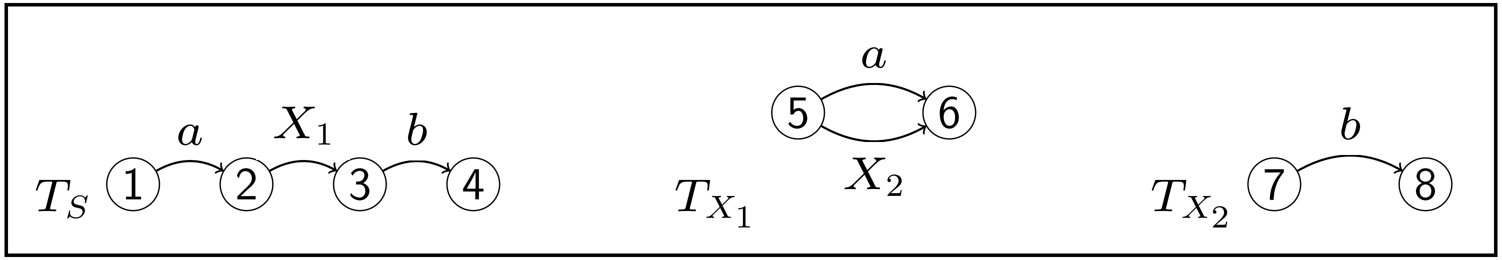
– $\mathbb{N}$ is a set of **non-terminals**; these serve as *pointers* to other FSAs

– $(T_\nu)_{\nu \in \mathbb{N}}$ – a family of FSAs with input alphabet $\Sigma \cup \mathbb{N}$

– S is the *root symbol*, $S \in \mathbb{N}$, and T_S is the *root FSA*

A string $x \in \Sigma^*$ is accepted by R if there is an accepting path in T_S such that recursively replacing every transition with the label $\nu \in \mathbb{N}$ by a path from T_ν leads to a path π^* such that $x = i[\pi^*]$.

RTN R : $\mathbb{N} = \{S, X_1, X_2\}$, $\Sigma = \{a, b\}$



Equivalent FSA: