Probabilistic FSAs, Weighted FSTs, and Weighted Logic

1 Probabilistic Automata

Definition 1 *A PFA is a tuple* $\mathcal{A} = \langle Q, \Sigma, \delta, I, F, P \rangle$ *where:*

Q	*is a finite set of states*
Σ	*is the alphabet*
$\delta \subseteq Q \times \Sigma \times Q$	*is a set of transitions*
$I : Q \to \mathbb{R}^+$	*are the initial-state probabilities*
$F : Q \to \mathbb{R}^+$	*are the final-state probabilities*
$P : \delta \to \mathbb{R}^+$	*are the transition probabilities*

I, F, and P are functions such that

$$\sum_{q \in Q} I(q) = 1$$

and

$$\forall q \in Q, F(q) + \sum_{a \in \Sigma, q' \in Q} P(q, a, q') = 1$$

Recall that FSA's can be thought of both as machines which *recognize* a given formal language, but they can also be thought of as machines which *generate* strings from the formal language. PFA's align more with the latter view.

Definition 2 *A PFA* $\mathcal{A} = \langle Q, \Sigma, \delta, I, F, P \rangle$ *is a deterministic PFA if:*

$$\exists q_0 \in Q, \textit{such that } I(q_o) = 1$$

$$\forall q \in Q, \forall a \in \Sigma, |\{q' : (q, a, q') \in \delta\}| \leq 1$$

What is the probability that a PFA $\mathcal{A}$ generates a string $x \in \Sigma^*$? First, define a path θ as

$$(s_0, x'_1, s_1, x'_2, s_2, \ldots, s_{k-1}, x'_k, s_k)$$

The probability of generating this path is

$$Pr_{\mathcal{A}}(\theta) = I(s_o) \cdot \left(\prod_{j=1}^{k} P(s_{j-1}, x'_j, s_j) \right) \cdot F(s_k)$$

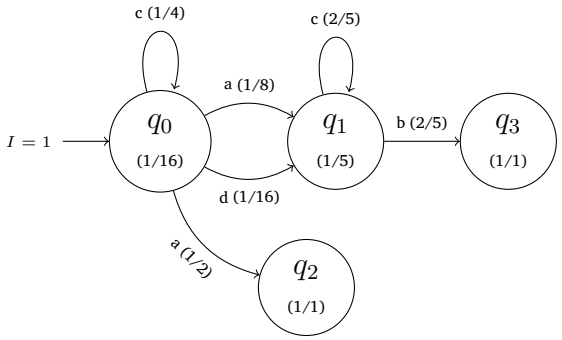
Definition 3 *A valid path in a PFA, $\mathcal{A}$ is a path for some $x \in \Sigma^*$ with probability greater than zero. The set of valid paths in $\mathcal{A}$ is denoted as Θ.*

Since there are potentially multiple valid paths for a string x in $\mathcal{A}$, we denote this as $\Theta_{\mathcal{A}}(x)$. With this, we can know say that the probability of generating string x with $\mathcal{A}$ is

$$Pr_{\mathcal{A}}(x) = \sum_{\theta \in \Theta_{\mathcal{A}}(x)} Pr_{\mathcal{A}}(\theta)$$

If $\sum_x Pr_{\mathcal{A}}(x) = 1$ then $\mathcal{A}$ defines a distribution $\mathcal{D}$ on Σ^*.

Let's discuss the following PFA in graphical form to ensure we all understand how these work.



How many valid paths are there for $x = accb$? How about for $x = a$? Does this PFA match the stipulations defined above? Is this PFA deterministic?

To end, I would like to leave you with the following two propositions. These are taken from Vidal et al. (2005a,b). If you are interested in what has been discussed so far, I highly recommend you reading these two papers carefully.

Proposition 1 *Given a PFA $\mathcal{A}$ with m transitions and $Pr_{\mathcal{A}}(\lambda) = 0$, there exists a Hidden Markov Model $\mathcal{M}$ with at most m states, such that $\mathcal{D}_{\mathcal{M}} = \mathcal{D}_{\mathcal{A}}$.*

Proposition 2 *Given an HMM $\mathcal{M}$ with n states, there exists a PFA with at most n states such that $\mathcal{D}_{\mathcal{A}} = \mathcal{D}_{\mathcal{M}}$.*

2 Weighted FSTs

Big Picture: Semirings are an algebraic structure. They are of interest to computational linguists because they allow for different output interpretations of the same input struc-

ture. Mohri (1997) showed how weighted finite state transducers (which require a specific semiring for interpretation) were useful for language and speech processing tasks. Goodman (1999) shortly after showed how different outputs of syntactic parsing could be described by the same algorithm by substituting the operations of different semirings. Roark and Sproat (2007) discuss semirings in their *Computational approaches to Morphology and Syntax* book and most recently Gorman and Sproat (2021) discuss how weighted finite state transducers form the backbone of their Python package `pynini` which is a wrapper for the more general `openfst` package.

My personal interest lies in the application of semirings to phonological theory. Heinz (Forthcoming) shows how to apply techniques from logic and model theory to analyzing phonological data. He includes a chapter on weighted logic which provides a way to account for the gradient application of certain phonological patterns. Weighted logic, as was the case for weighted transducers, requires the declaration of a specific semiring for interpretation. Another area of interest within phonological theory is how to best account for optional processes. These seem to require non-determinism, but it has recently been noticed that the semiring view may provide a way to have deterministic analyses of optional phonological processes. The specific semiring called the *Language* semiring (to be discussed below) is the crucial ingredient. This reframes the output of the phonological component of the grammar not as a single string, but as a set of strings. Because of this, it requires further choice between output components. In the simple case, this aligns exactly with abstract analyses of the phonetics-phonology interface that I have discussed in my dissertation and related papers (Nelson, 2024; Nelson and Heinz, in press; Nelson, to appear). I have additional papers in preparation to handle the not simple cases.

Semirings

This introduction follows Heinz (Forthcoming) which follows Droste and Gastin (2009). A semiring is a set $\mathcal{S}$ with two binary operations $\oplus$ and $\otimes$, and elements 0 and 1. All elements $x, y \in \mathcal{S}$ satisfy the following properties.

P1	$x \oplus y, x \otimes y \in \mathcal{S}$	(closure under $\oplus$ and $\otimes$)
P2	$x \oplus y = y \oplus x$	($\oplus$ is commutative)
P3	$0 \oplus x = x \oplus 0 = x$	(0 is the identity for $\oplus$)
P4	$1 \otimes x = x \otimes 1 = x$	(1 is the identity for $\otimes$)
P5	$0 \otimes x = x \otimes 0 = 0$	(0 is an annihilator for $\otimes$)
P6	$x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$	($\otimes$ right distributes over $\oplus$)

Some examples of semirings are given below.

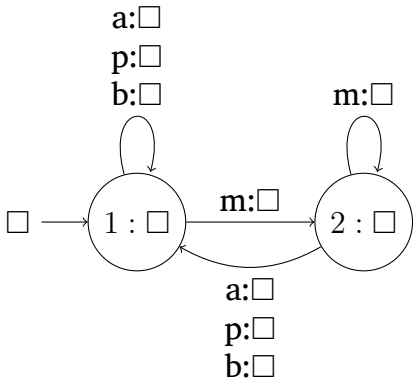
Name	$\mathcal{S}$	$\oplus$	$\otimes$	0	1
Boolean	$\{\texttt{TRUE}, \texttt{FALSE}\}$	$\vee$	$\wedge$	FALSE	TRUE
Finite Language	FIN	$\cup$	$\cdot$	$\emptyset$	$\{\lambda\}$
Probablity	$\mathbb{R}^+$	$+$	$\times$	0	1
Natural	$\mathbb{N}$	$+$	$\times$	0	1
Viterbi	$[0, 1]$	max	$\times$	0	1

Generalized Finite State Transducers

Semirings are required when we are dealing with non-determinism. Conceptually, $\otimes$ tells us to do as we follow a path while $\oplus$ tells us what to do if multiple paths result in the same output. I am primarily interested in *deterministic* computations. When dealing with determinism, we can take a substructure of a semiring called a **monoid**. A monoid $\mathfrak{M}$ consists of $\langle \mathcal{S}, e, \cdot \rangle$. Here, $\cdot$ is a binary operation and $e \in \mathcal{S}$ is an identity for that operation. We can generalized the notion of a deterministic finite state transducer with monoids.

Σ is the input alphabet/set of symbols
$\mathfrak{M}$ is a monoid
$\otimes \in \mathfrak{M}$ is a binary operation for combining output elements
Q is a set of states
$v_0 \in \mathcal{S}$ is the initial value
$q_0 \in Q$ is a single initial state that is a member of Q
δ is a transition function: $\Sigma \times Q \to v \in \mathcal{S} \times Q$
F is a final function: $Q \to v \in \mathcal{S}$

Suppose we wanted to formalize the process of post-nasal voicing. The following is a visual interpretation of a one-way deterministic transducer that would compute this (for a restricted input alphabet {a,b,p,m}). Notably here, all of the output symbols, as well as the input symbol and final symbols are missing. This is because these are the values determined by the monoid. We can use this same structure to compute many different things: whether the input is a valid string ($\langle \{\texttt{TRUE}, \texttt{FALSE}\}, \texttt{TRUE}, \wedge \rangle$), how many instances of a bad substring there are ($\langle \mathbb{N}, 0, + \rangle$), the weight of different inputs ($\langle \mathbb{R}, 1, \times \rangle$), what the singular output string should be ($\langle \Sigma^*, \lambda, \cdot \rangle$), what the set of output strings should be ($\langle \texttt{FIN}, \{\lambda\}, \cdot \rangle$).

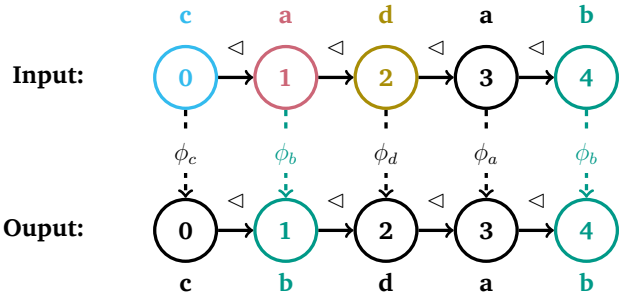


3 Weighted Logic

Let's review logical transductions very briefly. First, a **model signature** $\mathcal{S}$ is a collection of symbols for the functions, relations, and constants that are used to describe structures; *e.g.,* $\langle \lhd, \{R_\sigma \mid \sigma \in \Sigma\}\rangle$. An $\mathcal{S}$**-structure** A contains a set called the **domain**, as well as **denotations** for each symbol in $\mathcal{S}$. A **logical language** in first-order logic is defined by combining the symbols of first-order logic with a specific model signature $\mathcal{S}$. An interpretation of structure A in terms of structure B is a function denoted by a set of n formulas $\{\phi_i, \ldots, \phi_n\}$ where n is equal to the number of functions, relations, and constants in A's model signature, plus a domain formula, copy set, and licensing formula. A formula $\phi_P(x) \stackrel{\text{def}}{=} Q(x)$ denotes that domain element x has property P in the output structure only if it has property Q in the input structure. The general idea of logical transductions builds on the idea of interpretation, but was specifically lifted to account for string to string functions by computer scientists (**??**) and more recently phonological analysis (Heinz, Forthcoming).

Supposed we wanted to translate the rule $a \to b / c\_d$ into a logical transduction. We can do so with the following set of formulas.

$$\phi_a(x) \stackrel{\text{def}}{=} a(x) \wedge \neg\exists y, z[y \lhd x \lhd z \wedge c(y) \wedge d(z)]$$
$$\phi_b(x) \stackrel{\text{def}}{=} b(x) \vee (a(x) \wedge \exists y, z[y \lhd x \lhd z \wedge c(y) \wedge d(z)])$$
$$\phi_c(x) \stackrel{\text{def}}{=} c(x)$$
$$\phi_d(x) \stackrel{\text{def}}{=} d(x)$$



The model-theoretic approach also provides a way for identifying substructures with logic. Typically, these are boolean functions over strings: $f : \Sigma^* \to \{\texttt{TRUE}, \texttt{FALSE}\}$ These allow us to ask questions like does structure A contain the substring cad?

$$\varphi \stackrel{\text{def}}{=} \exists x[\exists y[y \lhd x \wedge c(y)] \wedge \exists z[x \lhd z \wedge d(z)] \wedge a(x) \wedge \texttt{TRUE}]$$

What I want to show you all is that it is possible to generalize the current approach by changing the type of the co-domain to any *semiring* S: $f : \Sigma^* \to \text{S}$.

Weighted logic is MSO logic where:

- $s \in \text{S}$ is an atomic formula
- Negation is only allowed in atomic formulas
- $\phi \wedge \psi$ is interpreted as $\phi \otimes \psi$
- $\phi \vee \psi$ is interpreted as $\phi \oplus \psi$
- $\forall x\phi$ is interpreted as $\phi(x_1) \otimes \phi(x_2) \otimes \cdots \otimes \phi(x_n) \;\; \forall x \in \mathcal{D}$
- $\exists x\phi$ is interpreted as $\phi(x_1) \oplus \phi(x_2) \oplus \cdots \oplus \phi(x_n) \;\; \forall x \in \mathcal{D}$
- ...

For more on the technical aspects, see Droste and Gastin (2009)[1]. in the meantime, let's look again at postnasal voicing. Below is a general formula where the boxes are temporary placeholders.

$$\varphi \stackrel{\text{def}}{=} (a(x) \wedge \square) \vee (b(x) \wedge \square) \vee (p(x) \wedge \square) \vee (m(x) \wedge \square) \vee (m(x) \wedge p(s(x)) \wedge \square)$$

Recall from above that we can switch out the conjunction and disjunction symbols for $\otimes$ and $\oplus$ symbols which are then interpreted in relation to a specific semiring.

[1] If you have a copy of Heinz (Forthcoming), there is also more information in chapters 4 and 6.

$$\varphi \stackrel{\text{def}}{=} (a(x) \otimes \square) \oplus (b(x) \otimes \square) \oplus (p(x) \otimes \square) \oplus (m(x) \otimes \square) \oplus (m(x) \otimes p(s(x)) \otimes \square)$$

If we wanted to use this formula to count how many times there was an instance of a voiceless stop after a nasal we can change the $\otimes$ and $\oplus$ symbols into regular $\times$ and $+$ while also filling the boxes with 0 for each subformula corresponding to a unigram, and 1 to the specific bigram formula that is only true when there is a voiceless stop after a nasal.

$$\varphi \stackrel{\text{def}}{=} (a(x) \times 0) + (b(x) \times 0) + (p(x) \times 0) + (m(x) \times 0) + (m(x) \times p(s(x)) \times 1)$$

Suppose we wanted to evaluate the string $a_1 m_2 p_3 a_4$ again. We would have to evaluate each index individually. Recall also that for this semiring $\texttt{TRUE} = 1$ and $\texttt{FALSE} = 0$. First we check domain element 1...

$$\begin{gathered}(a(1) \times 0) + (b(1) \times 0) + (p(1) \times 0) + (m(1) \times 0) + (m(1) \times p(s(1)) \times 1) \\ (\top \times 0) + (\bot \times 0) + (\bot \times 0) + (\bot \times 0) + (\bot \times \bot \times 1) \\ (1 \times 0) + (0 \times 0) + (0 \times 0) + (0 \times 0) + (0 \times 0 \times 1) \\ (0) + (0) + (0) + (0) + (0) \\ 0\end{gathered}$$

Then for domain element 2...

$$\begin{gathered}(a(2) \times 0) + (b(2) \times 0) + (p(2) \times 0) + (m(2) \times 0) + (m(2) \times p(s(2)) \times 1) \\ (\bot \times 0) + (\bot \times 0) + (\bot \times 0) + (\top \times 0) + (\top \times \top \times 1) \\ (0 \times 0) + (0 \times 0) + (0 \times 0) + (1 \times 0) + (1 \times 1 \times 1) \\ (0) + (0) + (0) + (0) + (1) \\ 1\end{gathered}$$

And so on. Since we're counting, we also want to add everything up. So ultimately we want to wrap our formula in an existential quantifier because this "adds" up the results of each index. What if instead, we wanted to use the finite language semiring? This will result in us mapping a single string to potentially multiple output strings. We update φ to look like this:

$$\varphi \stackrel{\text{def}}{=} (a(x) \cdot \{a\}) \cup (b(x) \cdot \{b\}) \cup (p(x) \cdot \{p\}) \cup (m(x) \cdot \{m\}) \cup (m(x) \cdot p(s(x)) \cdot \{b\})$$

Recall that "multiplication here is *language concatenation* which is defined as $XY = \{xy \mid x \in X, y \in Y\}$. Importantly, $X\emptyset = \emptyset X = \emptyset$ and $X\{\lambda\} = \{\lambda\}X = X$. This is why $\emptyset$ is the equivalent to `FALSE` and $\{\lambda\}$ is the equivalent to `TRUE`. We play the same game, checking domain element 1...

$$(a(1) \cdot \{a\}) \cup (b(1) \cdot \{b\}) \cup (p(1) \cdot \{p\}) \cup (m(1) \cdot \{m\}) \cup (m(1) \cdot p(s(1)) \cdot \{b\})$$
$$(\top \cdot \{a\}) \cup (\bot \cdot \{b\}) \cup (\bot \cdot \{p\}) \cup (\bot \cdot \{m\}) \cup (\bot \cdot \bot \cdot \{b\})$$
$$(\{\lambda\} \cdot \{a\}) \cup (\emptyset \cdot \{b\}) \cup (\emptyset \cdot \{p\}) \cup (\emptyset \cdot \{m\}) \cup (\emptyset \cdot \emptyset \cdot \{b\})$$
$$(\{a\}) \cup (\emptyset) \cup (\emptyset) \cup (\emptyset) \cup (\emptyset)$$
$$\{a\}$$

Then we check domain element 2...

$$(a(2) \cdot \{a\}) \cup (b(2) \cdot \{b\}) \cup (p(2) \cdot \{p\}) \cup (m(2) \cdot \{m\}) \cup (m(2) \cdot p(s(2)) \cdot \{b\})$$
$$(\bot \cdot \{a\}) \cup (\bot \cdot \{b\}) \cup (\bot \cdot \{p\}) \cup (\top \cdot \{m\}) \cup (\top \cdot \top \cdot \{b\})$$
$$(\emptyset \cdot \{a\}) \cup (\emptyset \cdot \{b\}) \cup (\emptyset \cdot \{p\}) \cup (\{\lambda\} \cdot \{m\}) \cup (\{\lambda\} \cdot \{\lambda\} \cdot \{b\})$$
$$(\emptyset) \cup (\emptyset) \cup (\emptyset) \cup (\{m\}) \cup (\{b\})$$
$$\{m, b\}$$

Uh oh, we ran into a problem...for counting mp sequences it turns out not to matter, but for the optional mapping, we want the thing that has multiple outputs to be x in our subformula, and the adjacent trigger to be found using predecessor or succesorr. This is an easy fix. Instead of the last subformula being $m(x) \wedge p(s(x))$, we can just change it to $p(x) \wedge m(\mathtt{p}(x))$. Everything else works out exactly the same. For this analysis, we want to wrap our formula in the universal quantifier because we want to "multiply" aka do language concatenation across each domain element. Nonetheless, the core formula is exactly the same for counting as it is for mapping to multiple strings. This echoes the WFST analysis.

References

Droste, M. and Gastin, P. (2009). Weighted automata and weighted logics. In *Handbook of weighted automata*, pages 175–211. Springer.

Goodman, J. (1999). Semiring parsing. *Computational Linguistics*, 25(4):573–606.

Gorman, K. and Sproat, R. (2021). *Finite-State Text Processing*. Morgan & Claypool Publishers.

Heinz, J. (Forthcoming). *Doing Computational Phonology*. Oxford University Press.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational linguistics*, 23(2):269–311.

Nelson, S. (2024). *The Computational Structure of Phonological and Phonetic Knowledge*. PhD thesis, Stony Brook University.

Nelson, S. (to appear). Optionality and the phonetics-phonology interface. In *Proceedings of WCCFL 42*.

Nelson, S. and Heinz, J. (in press). The blueprint model of production. *Phonology*.

Roark, B. and Sproat, R. (2007). *Computational approaches to morphology and syntax*, volume 4. Oxford University Press.

Vidal, E., Thollard, F., De La Higuera, C., Casacuberta, F., and Carrasco, R. C. (2005a). Probabilistic finite-state machines-part i. *IEEE transactions on pattern analysis and machine intelligence*, 27(7):1013–1025.

Vidal, E., Thollard, F., De La Higuera, C., Casacuberta, F., and Carrasco, R. C. (2005b). Probabilistic finite-state machines-part ii. *IEEE transactions on pattern analysis and machine intelligence*, 27(7):1026–1039.