3.3 Input Strictly k-Local Transductions

Tracing the paths through a given deterministic finite-state automaton also helps us understand how transductions can be learned. A string-to-string transduction is a function from Σ^* to Δ^*.

Each DFA describes a class of transductions as follows. Each transition $\delta(q, a)$ is associated with an output string θ_{qa}. Each state is also associated with an output string $\theta_{q\ltimes}$, and the initial state is also associated with an output string $\theta_{\rtimes}$. The strings associated with each transitions and each state are the *parameters* of the model the DFA describes.

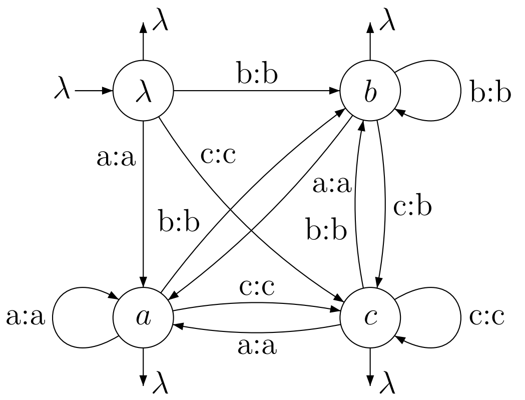
We define a function π which describes the computational "process" and "path" string $w \in \Sigma^*$ takes from any state $q \in Q$ with any initial string value $v \in \Delta^*$. The operator $(\cdot)$ refers to concatenation.

$$\begin{aligned}\pi(q, \lambda, v) &= v \cdot \theta_{q\ltimes} \\ \pi(q, wa, v) &= \pi(\delta(q, a), w, v \cdot \theta_{qa})\end{aligned} \tag{6}$$

Then the function such a string-weighted DFA A describes is given by the equation below.

$$f_A(w) = \pi(q_0, w, \theta_{\rtimes}) \tag{7}$$

Figure 4 shows two Input Strictly 2-Local functions with $\Sigma = \Delta = \{a, b, c\}$. The first function describes "progressive b-assimilation." As a rewrite rule, this process would be expressed as $c \rightarrow b/b\_\_$. The second function describes "regressive b-assimilation." As a rewrite rule, this process would be expressed as $c \rightarrow b/\_\_b$.



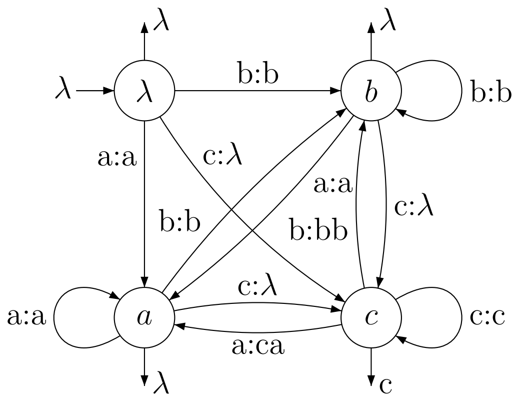


Figure 4: 2-ISL transducers for progressive b-assimilation (left) and regressive b-assimilation (right).

The identification in the limit paradigm for positive data natrually yields a definitions transduction learning. A positive presentation is now example transductions, which are input-output pairs $(w, f(w))$.

More precisely, a **positive presentation** of a string-to-string function f is a function $\varphi : \mathbb{N} \rightarrow f$ such that φ is onto. This means for every input-output pair $(w, f(w))$ defined by f, there is some $n \in \mathbb{N}$ such that $\varphi(n) = (w, f(w))$.

Definition 7 (Identification in the limit from positive data (function version))**.**

Algorithm A *identifies in the limit from positive data* a class of string-to-string functions C provided

```
for all functions $f \in C$,
    for all positive presentations $\varphi$ of $f$,
        there is some number $n \in \mathbb{N}$ such that
            for all $m > n$,
            • the program output by $A$ on $\varphi\langle m\rangle$ is the same as the the program
              output by $A$ on $\varphi\langle n\rangle$, and
            • the program output by $A$ on $\varphi\langle m\rangle$ which takes any string $w \in \Sigma^*$
              for which $f$ is defined as input and returns $f(w)$ as output.
```

The algorithm SOSFIA (Structured Onward Subsequential Function Inference Algorithm) (Jardine *et al.*, 2014) provably identifies the k-ISL functions in the limit from positive data.

Here is a summary of how SOSFIA works with illustrations by examples. Given a sample S of input-output pairs, SOSFIA calculates the *common output* (`common_out`) of every prefix of any input string. The common output of an input prefix u is the longest prefix common to all the output strings whose corresponding input strings have prefix u. This *longest common prefix* is denoted `lcp`.

SOSFIA then uses these common outputs to calculate the *minimal change* (`min_change`) each letter introduces to the output string. These minimal changes are the parameter values (the outputs associated with the transitions). Minimal change is calculated using "left division." This operation "strips away" a prefix of a string. Formally, whenever $w = uv$ then $u^{-1}w = v$. We say "the left division of w by u equals v."

Formal definitions of `common_out` and `min_change` from Jardine *et al.* (2014, p. 101).

Definition 8. The *common output* of an input prefix w in a sample $S \subset \Sigma^* \times \Delta^*$ for t is the `lcp` of all $t(wv)$ that are in S: $\texttt{common\_out}_S(w) = \texttt{lcp}(\{u \in \Sigma^* \mid \exists v \text{ s.t. } (wv, u) \in S\})$

Definition 9. The *minimal change in the output* in $S \subset \Sigma^* \times \Delta^*$ from w to $w\sigma$ is:

$$\texttt{min\_change}_S(\sigma, w) = \begin{cases} \texttt{common\_out}_S(\sigma) & \text{if } w = \lambda \\ \texttt{common\_out}_S(w)^{-1}\texttt{common\_out}_S(w\sigma) & \text{otherwise} \end{cases}$$

Consider progressive b-assimilation and let the sample S be as shown.

$$S = \{(aa, aa), (ab, ab), (ac, ac), (ba, ba), (bb, bb), (bc, bb), (ca, ca), (cb, cb), (cc, cc)\}$$

The input prefixes in this sample are $S_i = \{\lambda, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc\}$. So we need to calculate the longest common prefix of all the outputs associated with each string. Here they are.

$\texttt{common\_out}_S(\lambda)$	$=$	$\texttt{lcp}(f(S_i))$	$=$	λ
$\texttt{common\_out}_S(a)$	$=$	$\texttt{lcp}(f(a), f(aa), f(ab), f(ac))$	$=$	a
$\texttt{common\_out}_S(b)$	$=$	$\texttt{lcp}(f(b), f(ba), f(bb), f(bc))$	$=$	b
$\texttt{common\_out}_S(c)$	$=$	$\texttt{lcp}(f(c), f(ca), f(cb), f(cc))$	$=$	c
$\texttt{common\_out}_S(aa)$	$=$	$\texttt{lcp}(f(aa))$	$=$	aa
$\texttt{common\_out}_S(ab)$	$=$	$\texttt{lcp}(f(ab))$	$=$	ab
$\texttt{common\_out}_S(ac)$	$=$	$\texttt{lcp}(f(ac))$	$=$	ac
$\texttt{common\_out}_S(ba)$	$=$	$\texttt{lcp}(f(ba))$	$=$	ba
$\texttt{common\_out}_S(bb)$	$=$	$\texttt{lcp}(f(bb))$	$=$	bb
$\texttt{common\_out}_S(bc)$	$=$	$\texttt{lcp}(f(bc))$	$=$	bb
$\texttt{common\_out}_S(ca)$	$=$	$\texttt{lcp}(f(ca))$	$=$	ca
$\texttt{common\_out}_S(cb)$	$=$	$\texttt{lcp}(f(cb))$	$=$	cb
$\texttt{common\_out}_S(cc)$	$=$	$\texttt{lcp}(f(cc))$	$=$	cc

With the `common_out` values we can calculate the minimal changes.

$\texttt{min\_change}_S(a, \lambda)$	$=$	$\texttt{common\_out}_S(a)$			$=$	a	
$\texttt{min\_change}_S(b, \lambda)$	$=$	$\texttt{common\_out}_S(b)$			$=$	b	
$\texttt{min\_change}_S(c, \lambda)$	$=$	$\texttt{common\_out}_S(c)$			$=$	c	
$\texttt{min\_change}_S(a, a)$	$=$	$\texttt{common\_out}_S(a)^{-1}\texttt{common\_out}_S(aa)$	$=$	$a^{-1}aa$	$=$	a	
$\texttt{min\_change}_S(b, a)$	$=$	$\texttt{common\_out}_S(a)^{-1}\texttt{common\_out}_S(ab)$	$=$	$a^{-1}ab$	$=$	b	
$\texttt{min\_change}_S(c, a)$	$=$	$\texttt{common\_out}_S(a)^{-1}\texttt{common\_out}_S(ac)$	$=$	$a^{-1}ac$	$=$	c	
$\texttt{min\_change}_S(a, b)$	$=$	$\texttt{common\_out}_S(b)^{-1}\texttt{common\_out}_S(ba)$	$=$	$b^{-1}ba$	$=$	a	
$\texttt{min\_change}_S(b, b)$	$=$	$\texttt{common\_out}_S(b)^{-1}\texttt{common\_out}_S(bb)$	$=$	$b^{-1}bb$	$=$	b	
$\texttt{min\_change}_S(c, b)$	$=$	$\texttt{common\_out}_S(b)^{-1}\texttt{common\_out}_S(bc)$	$=$	$b^{-1}bb$	$=$	b	(!!)
$\texttt{min\_change}_S(a, c)$	$=$	$\texttt{common\_out}_S(c)^{-1}\texttt{common\_out}_S(ca)$	$=$	$c^{-1}ca$	$=$	a	
$\texttt{min\_change}_S(b, c)$	$=$	$\texttt{common\_out}_S(c)^{-1}\texttt{common\_out}_S(cb)$	$=$	$c^{-1}cb$	$=$	b	
$\texttt{min\_change}_S(c, c)$	$=$	$\texttt{common\_out}_S(c)^{-1}\texttt{common\_out}_S(cc)$	$=$	$c^{-1}cc$	$=$	c	

The minimal change letter σ with string w gives us the output string at the state $\texttt{suff}_{k-1}(w)$ for the transition labeled σ. Above, we would have $\texttt{min\_change}_S(\sigma, q) = \theta_{q\sigma}$.

Now consider regressive b-assimilation and let the sample S be as shown.

$$S = \left\{ \begin{array}{l} (aa,aa),(ab,ab),(ac,ac),(ba,ba),(bb,bb),(bc,bc),(ca,ca),(cb,bb),(cc,cc), \\ (aca,aca),(acb,acb),(bca,bca),(bcb,bbb),(cca,cca),(ccb,cbb) \end{array} \right\}$$

As before, the input prefixes in this sample are $S_i = \{\lambda, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc\}$. So we need to calculate the longest common prefix of all the outputs associated with each string. Here they are.

$\texttt{common\_out}_S(\lambda)$	=	$\texttt{lcp}(f(S_i))$	=	λ	
$\texttt{common\_out}_S(a)$	=	$\texttt{lcp}(f(a), f(aa), f(ab), \ldots)$	=	a	
$\texttt{common\_out}_S(b)$	=	$\texttt{lcp}(f(b), f(ba), f(bb), \ldots)$	=	b	
$\texttt{common\_out}_S(c)$	=	$\texttt{lcp}(f(c), f(ca), f(cb), \ldots)$	=	λ	(!!)
$\texttt{common\_out}_S(aa)$	=	$\texttt{lcp}(f(aa))$	=	aa	
$\texttt{common\_out}_S(ab)$	=	$\texttt{lcp}(f(ab))$	=	ab	
$\texttt{common\_out}_S(ac)$	=	$\texttt{lcp}(f(ac), f(aca), f(acb))$	=	a	(!!)
$\texttt{common\_out}_S(ba)$	=	$\texttt{lcp}(f(ba))$	=	ba	
$\texttt{common\_out}_S(bb)$	=	$\texttt{lcp}(f(bb))$	=	bb	
$\texttt{common\_out}_S(bc)$	=	$\texttt{lcp}(f(bc), f(bca), f(bcb))$	=	b	(!!)
$\texttt{common\_out}_S(ca)$	=	$\texttt{lcp}(f(ca))$	=	ca	
$\texttt{common\_out}_S(cb)$	=	$\texttt{lcp}(f(cb))$	=	bb	
$\texttt{common\_out}_S(cc)$	=	$\texttt{lcp}(f(cc), f(cca), f(ccb))$	=	c	(!!)

With the $\texttt{common\_out}$ values we can calculate the minimal changes.

$\texttt{min\_change}_S(a, \lambda)$	=	$\texttt{common\_out}_S(a)$			=	a
$\texttt{min\_change}_S(b, \lambda)$	=	$\texttt{common\_out}_S(b)$			=	b
$\texttt{min\_change}_S(c, \lambda)$	=	$\texttt{common\_out}_S(c)$			=	c
$\texttt{min\_change}_S(a, a)$	=	$\texttt{common\_out}_S(a)^{-1}\texttt{common\_out}_S(aa)$	=	$a^{-1}aa$	=	a
$\texttt{min\_change}_S(b, a)$	=	$\texttt{common\_out}_S(a)^{-1}\texttt{common\_out}_S(ab)$	=	$a^{-1}ab$	=	b
$\texttt{min\_change}_S(c, a)$	=	$\texttt{common\_out}_S(a)^{-1}\texttt{common\_out}_S(ac)$	=	$a^{-1}a$	=	λ
$\texttt{min\_change}_S(a, b)$	=	$\texttt{common\_out}_S(b)^{-1}\texttt{common\_out}_S(ba)$	=	$b^{-1}ba$	=	a
$\texttt{min\_change}_S(b, b)$	=	$\texttt{common\_out}_S(b)^{-1}\texttt{common\_out}_S(bb)$	=	$b^{-1}bb$	=	b
$\texttt{min\_change}_S(c, b)$	=	$\texttt{common\_out}_S(b)^{-1}\texttt{common\_out}_S(bc)$	=	$b^{-1}b$	=	λ
$\texttt{min\_change}_S(a, c)$	=	$\texttt{common\_out}_S(c)^{-1}\texttt{common\_out}_S(ca)$	=	$\lambda^{-1}ca$	=	ca
$\texttt{min\_change}_S(b, c)$	=	$\texttt{common\_out}_S(c)^{-1}\texttt{common\_out}_S(cb)$	=	$\lambda^{-1}bb$	=	bb
$\texttt{min\_change}_S(c, c)$	=	$\texttt{common\_out}_S(c)^{-1}\texttt{common\_out}_S(cc)$	=	$\lambda^{-1}c$	=	c

Again, we see that $\texttt{min\_change}_S(\sigma, q) = \theta_{q\sigma}$.

Readers are referred to the paper for full details on SOSFIA and that it provably identifies in the limit the class of k-ISL functions.

3.4 Output Strictly k-Local Transductions

Chandlee *et al.* (2015) prove a similar algorithm for inferring k-OSL functions.

3.5 Generalizing to any DFA

The aforementioned strategies hold for *any* deterministic finite-state automata. In other words, each deterministic finite state machine defines a class of stringsets, a class of stochastic stringsets, and a class of transductions, and each class can be learned with the methods described above.

Heinz and Rogers (2013) establish this for Boolean case. For stochastic stringsets, this result was known much earlier. Jardine *et al.* (2014) establish this for "Input" based transductions. For output-based transductions, the theorems Chandlee *et al.* (2015) do not address this general case, but the same techniques apply there as well.

To my knowledge, these results have not yet been ported to tree automata.

References

Chandlee, Jane. 2014. Strictly local phonological processes. Doctoral dissertation, The University of Delaware.

Chandlee, Jane, Rémi Eyraud, and Jeffrey Heinz. 2014. Learning strictly local subsequential functions. *Transactions of the Association for Computational Linguistics* 2:491–503.

Chandlee, Jane, Rémi Eyraud, and Jeffrey Heinz. 2015. Output strictly local functions. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, edited by Marco Kuhlmann, Makoto Kanazawa, and Gregory M. Kobele, 112–125. Chicago, USA.

Chandlee, Jane, and Jeffrey Heinz. Forthcoming. Strictly local phonological processes. *Linguistic Inquiry* .

Chandlee, Jane, Jeffrey Heinz, and Adam Jardine. To appear. Input strictly local opaque maps. *Phonology* .

Heinz, Jeffrey, and James Rogers. 2013. Learning subregular classes of languages with factored deterministic automata. In *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, edited by Andras Kornai and Marco Kuhlmann, 64–71. Sofia, Bulgaria: Association for Computational Linguistics.

Jardine, Adam, Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2014. Very efficient learning of structured classes of subsequential functions from positive data. In *Proceedings of the Twelfth International Conference on Grammatical Inference (ICGI 2014)*, edited by Alexander Clark, Makoto Kanazawa, and Ryo Yoshinaka, vol. 34, 94–108. JMLR: Workshop and Conference Proceedings.

Vidal, Enrique, Franck Thollard, Colin de la Higuera, Francisco Casacuberta, and Rafael C. Carrasco. 2005a. Probabilistic finite-state machines-part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27:1013–1025.

Vidal, Enrique, Frank Thollard, Colin de la Higuera, Francisco Casacuberta, and Rafael C. Carrasco. 2005b. Probabilistic finite-state machines-part II. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27:1026–1039.