# More efficient learning of strictly 2-local tier-based functions

Phillip Burness and Kevin McMullin
University of Ottawa, Canada

Mathematics of Language 17
December 13, 2021

# Overview

- Background
  - Notation
  - Tier-based Strictly Local functions
  - Previous learning results
- Learning Part 1: Estimating $f^p$
  - Previous approach
  - Current approach
- Learning Part 2: Identifying the tier
  - Previous approach
  - Current approach
- Discussion

# Notation

- The empty string is $\lambda$
- String concatenation is indicated with $\cdot$
  - $ab \cdot b = abb$
  - Omitted when context permits
- The exponent $-1$ represents string subtraction
  - $(ab)^{-1} \cdot abb = b$
- $\Sigma$ and $\Delta$ are the input and output alphabets
- $\Sigma^*$ is all strings of any length made using elements of $\Sigma$
- The *longest common prefix* of a stringset $S$ is $\texttt{lcp}(S)$
  - $\texttt{lcp}(\{a, aa, ab, aaa\}) = a$

# Notation

- For function $f$, its corresponding *prefix function* ($f^p$) is
  - $f^p(w) = \texttt{lcp}(f(w\Sigma^*))$
  - For example, if $f(w) = w \cdot a$ then $f^p(w) = w$
- The *contribution* of $x \in \Sigma^*$ given $w \in \Sigma^*$ with respect to the function $f$ is
  - $\texttt{cont}_f(x, w) = f^p(w)^{-1} \cdot f^p(wx)$
  - Essentially, the part of $f^p(wx)$ directly attributable to $x$
- The *contribution* of $\ltimes \notin \Sigma$ given $w \in \Sigma^*$ with respect to the function $f$ is
  - $\texttt{cont}_f(\ltimes, w) = f^p(w)^{-1} \cdot f(w)$
  - Essentially, the part of $f(w)$ that can't be directly attributed to input material

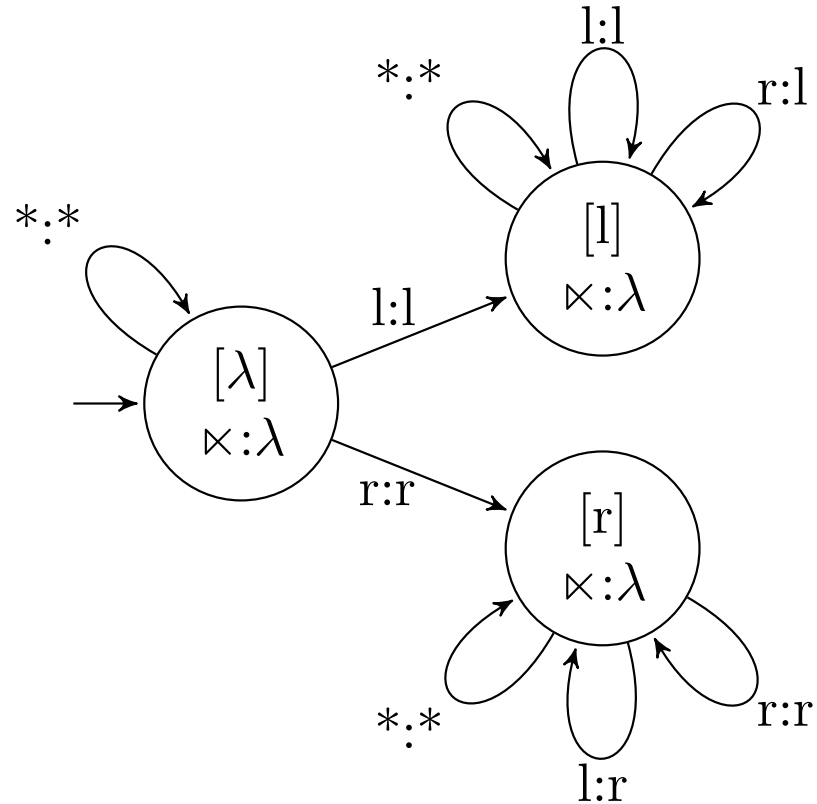
# Tier-based Strictly Local functions

- Functional analogues to the Tier-based Strictly Local (TSL) languages of Heinz et al. (2011)
- A *tier* is a subset of a given alphabet, containing only the "relevant" members
- Tiers let us create long-distance dependencies between tier elements by skipping over non-tier elements
- Given a tier $T$ and a string $w$, the tier projection $\pi_T(w)$ is $w$ with all non-members of $T$ removed
  - Given $T = \{l, r\}$, $\pi_T(lokiral) = lrl$
- We write $\texttt{suff}^n_T(w)$ for the $n$-long suffix of $\pi_T(w)$

# Tier-based Strictly Local functions

- A function is Input Tier-based Strictly k-Local ($\text{ITSL}_k$) iff there is a tier $T \subseteq \Sigma$ such that for all $x \in \Sigma^*$
  - $\texttt{suff}_T^{k-1}(w_1) = \texttt{suff}_T^{k-1}(w_2) \implies \texttt{cont}_f(x, w_1) = \texttt{cont}_f(x, w_2)$
- A function is Output Tier-based Strictly k-Local ($\text{OTSL}_k$) iff there is a tier $T \subseteq \Delta$ such that for all $x \in \Sigma^*$
  - $\texttt{suff}_T^{k-1}(f^p(w_1)) = \texttt{suff}_T^{k-1}(f^p(w_2)) \implies \texttt{cont}_f(x, w_1) = \texttt{cont}_f(x, w_2)$
- In other words, transformations depend entirely on the $k-1$ most recent input or output tier elements
- TSL functions can model many long-distance phonological processes such as vowel and consonant harmony (Andersson et al., 2020; Burness et al., 2021)

# TSL functions




OTSL$_2$ function for a simple pattern of liquid harmony

- /l/ → [r] / [r..._]
- /r/ → [l] / [l..._]

# Previous learning results

- Let $n$ be the size of the input data sample
- Strictly Local functions (of which TSL functions are a superclass) can be learned in $\mathcal{O}(n^2)$ time
  - Input Strictly Local Function Inference Algorithm (ISLFIA: Chandlee et al. 2014)
  - Output Strictly Local Function Inference Algorithm (OSLFIA: Chandlee et al. 2015)
- Subsequential functions (of which TSL functions are a subclass) can be learned in $\mathcal{O}(n^3)$ time
  - Onward Subsequential Transducer Inference Algorithm (OSTIA: Oncina et al. 1993)

# Previous learning results

- *When the tier is known*, TSL functions can be learned in $\mathcal{O}(n^2)$ time (Burness and McMullin, 2019)
- Identifying the tier is possible when $k = 2$ (Burness and McMullin, 2019)
- The method proposed by Burness and McMullin (2019) for this takes $\mathcal{O}(n^5)$ time
- We identified the bottlenecks responsible for the high time complexity, and will show how they can be eliminated to make tier identification run in $\mathcal{O}(n^2)$ time.

# Previous approach

- Our improved algorithm follows the overall architecture of the one in Burness and McMullin (2019)
- As an example target function, we will use symmetric sibilant harmony
  - $\Sigma = \{\text{s}, \text{ʃ}, \text{a}\}$
  - $\Delta = \{\text{s}, \text{ʃ}, \text{a}\}$
  - /s/ $\rightarrow$ [ʃ] if preceded by [ʃ] at any distance
  - /ʃ/ $\rightarrow$ [s] if preceded by [s] at any distance
  - $S = \{(w\ltimes, f(w)) \mid w \in \Sigma^{\leq 4}\}$
- The first thing that must be done: determine $f^p(w)$ for as many inputs $w$ as possible.

# Previous approach

- Start by adding all the input prefixes in $S$ to a set $X$
  - (sa⋉, sa) adds 'λ', 's', 'sa', and 'sa⋉' to $X$
- Then, for each $x \in X$ we check whether $x\sigma$ is also in $X$ for each $\sigma \in \Sigma \cup \{\ltimes\}$
- If so, we can calculate $f^p(x)$, so we add $x$ to another set $Y$.
  - For 's' $\in X$
    - 's⋉', 'sa', 'ss', 'sʃ'$\in X$
    - 's' is thus added to $Y$
- For each $y \in Y$, we determine $f^p(y) = \texttt{lcp}(\{u \mid (w, u) \in S$ and $y$ is a prefix of $w\})$
- This compiles a set $Z$ of pairs $(x, f^p(x))$ that is essentially a "snapshot" of $f^p$

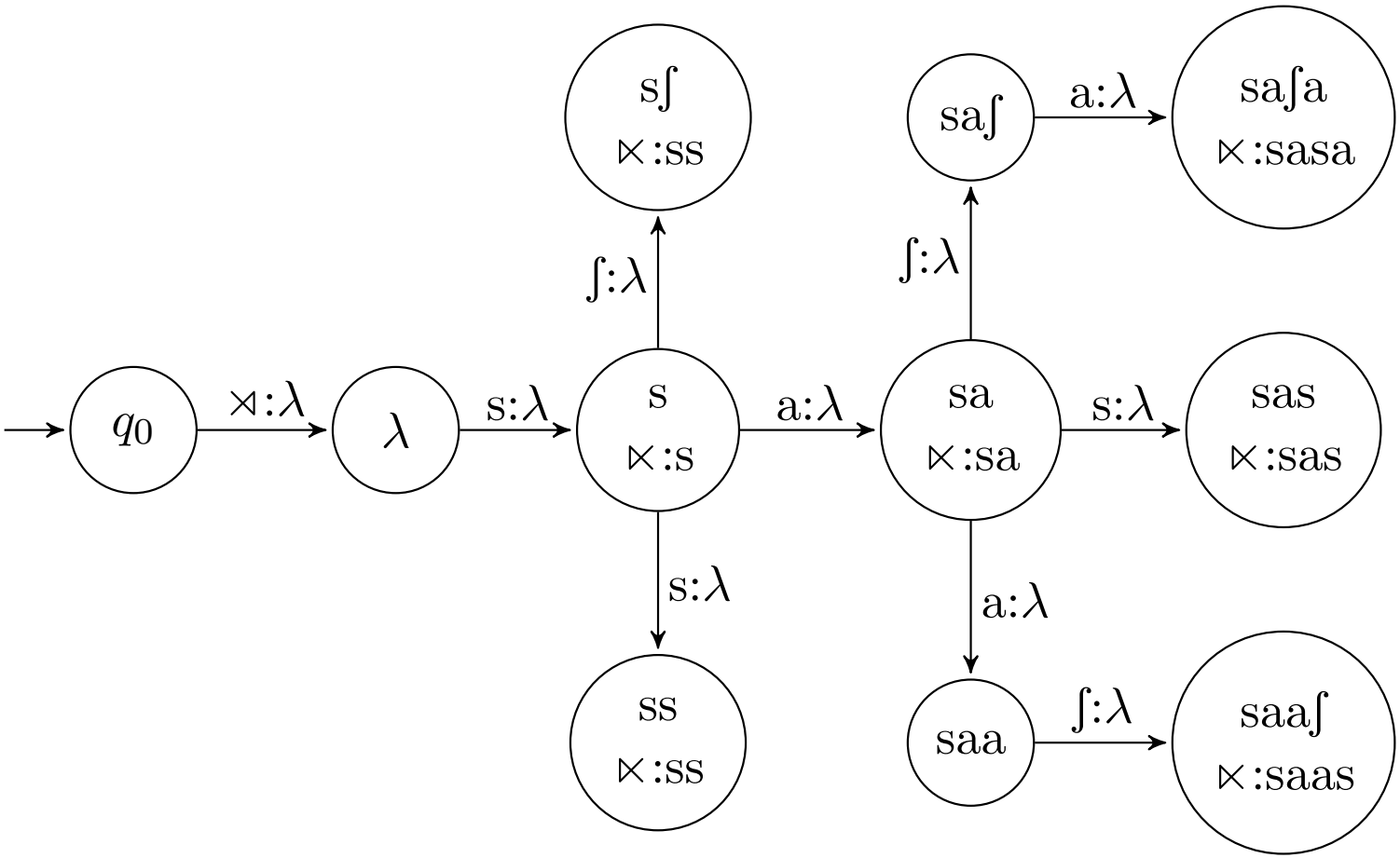
# Previous approach

- The process is inefficient because we are effectively performing nested reads of the sample
- Let $l$ be the number of training instances
  Let $m$ be the length of the longest input string
  Let $n$ be the summed length of the input strings
  - There can be up to $lm$ prefixes
  - For each, we calculate an `lcp` which can take up to $nm$ steps
  - Overall we have $\mathcal{O}(lm \cdot nm)$
- The same result can be obtained without nested reads by using a Prefix Tree Transducer (PTT)

# Prefix Tree Transducers

- A PTT corresponding to a sample generates all and only the pairs in the sample
- In their base form, they wait until the end of the input string is reached before outputting anything.
- The next slide shows the base PTT for the sample:
  - (s, s)
  - (ss, ss)
  - (sʃ, ss)
  - (sa, sa)
  - (sas, sas)
  - (saʃa, sasa)
  - (saaʃ, saas)

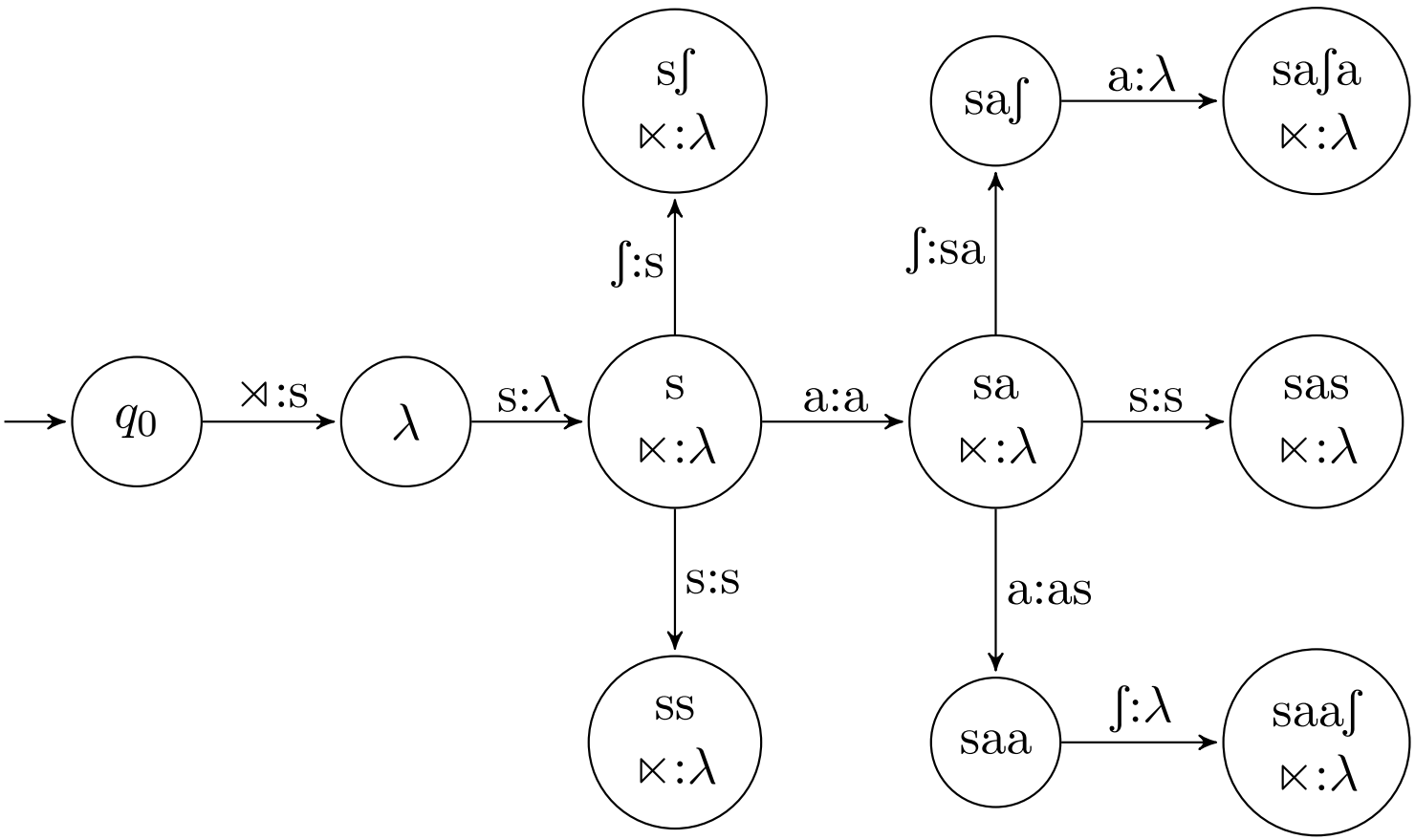
# Prefix Tree Transducers



# Onwarding

- A base PTT can be made *onward* so that it produces as much output as soon as possible
- For details on how to make a PTT onward, see chapter 18 of de la Higuera (2010)
- OSTIA and the ISLFIA (but not the OSLFIA) manipulate onward PTTs, merging their states until convergence
- For tier learning, we will use an onward PTT as a sort of oracle, consulting it for essential information
- The next slide shows the onward version of the prior PTT

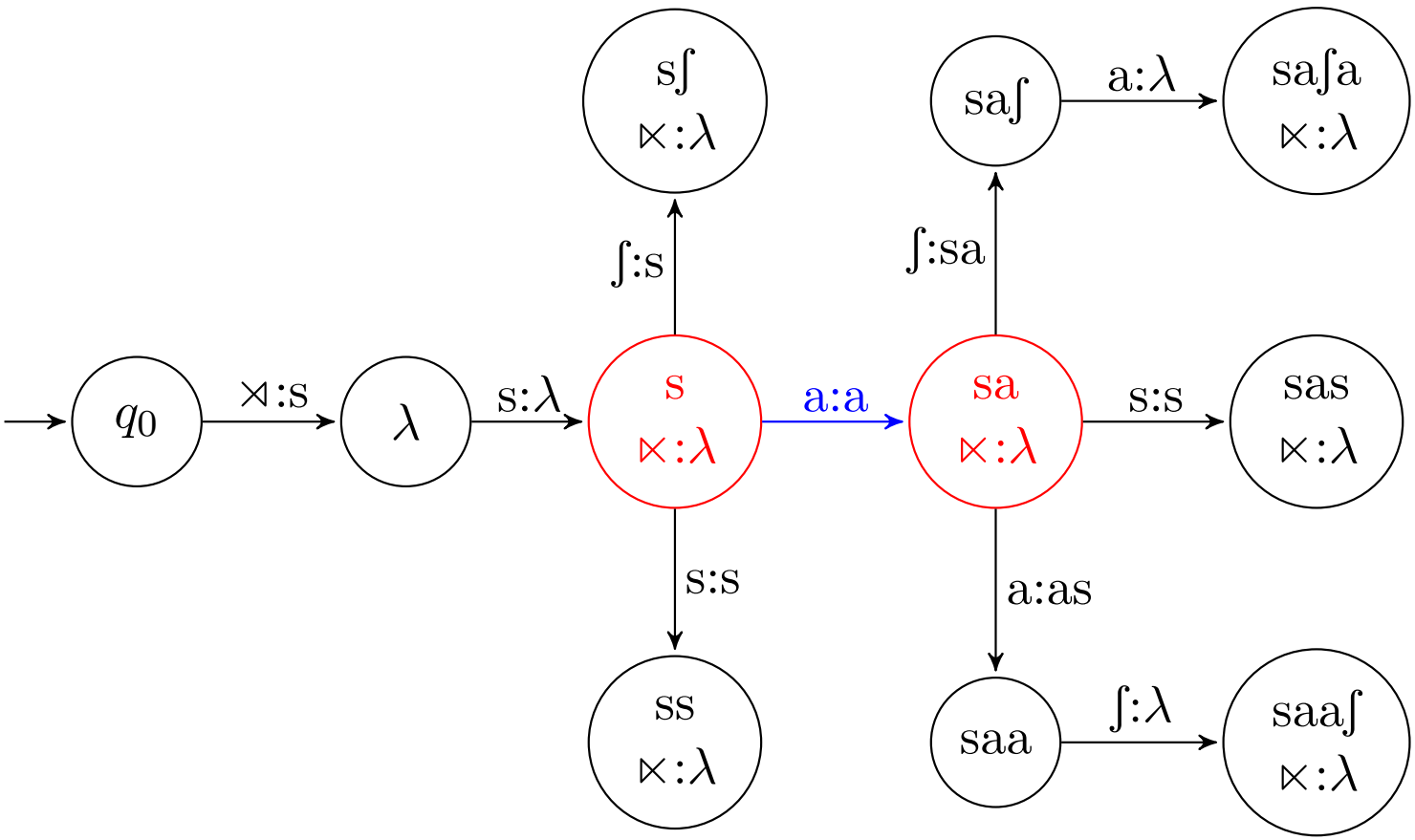
# Onwarding


$q_0$
⋊:s
λ
s:λ
s
⋉:λ
a:a
sa
⋉:λ
s:s
sas
⋉:λ
ʃ:s
sʃ
⋉:λ
s:s
ss
⋉:λ
ʃ:sa
saʃ
a:λ
saʃa
⋉:λ
a:as
saa
ʃ:λ
saaʃ
⋉:λ

# Current approach

- The key insight with regards to onward PTTs:
  - Given state $q$, if there is an outgoing transition for every possible input element (including word-end symbol $\ltimes$), then the PTT will have produced $f^p(q)$ upon landing in $q$
  - We call such states *supported states*
- Equally important corollaries:
  - The final transition out of supported $q$ produces $\texttt{cont}_f(\ltimes, q)$
  - A transition from supported $q$ to supported $q'$ for $\sigma$ produces $\texttt{cont}_f(\sigma, q)$

# Current approach



# Current approach

- We can compile the set $Z$ of pairs $(x, f^p(x))$ by sending the sample once through its onward PTT
- At each reading step, we check whether we are in a supported state.
- If yes, we add the corresponding pair to $Z$
- Onwarding the PTT takes time quadratic in the size of the sample, as does the just described process of checking the sample against the PTT
  - While $\Sigma$ is technically a constant for the learning problem, the runtime is linear in $|\Sigma|$
- Using a PTT reduces time complexity from $n^4$ to $n^2$

# Strategy overview

- Property 1: When a given $\text{TSL}_2$ function can be described using tier $A$ and using tier $B$, then it can also be described using tier $C = A \cup B$
  - Implies the existence of a largest *canonical* tier which properly contains any other tier that can be used
- Property 2: If we try to describe a $\text{TSL}_2$ function with a strict superset of its canonical tier, there will be evidence that a superfluous member can never be on any tier
  - There will never be such evidence for members of the canonical tier
- Consequence: We can start with the whole alphabet as the tier, and whittle it down as we find relevant evidence

# Previous approach

- We use the set $Z$ constructed by the previous portion of the algorithm (the "snapshot" of $f^p$)
- We cycle through the current tier hypothesis $H$ until all its elements get added to an auxiliary set $K$ (for "keep")
- To check whether $t \in H$ can be added to $K$
  - Gather all $(x, y) \in Z$ such that $\texttt{suff}^1_H(y) = t$
  - For each collected pair:
    - Check whether $(x\sigma, y') \in Z$ for each $\sigma \in \Sigma$
    - If so, calculate $\texttt{cont}_f(\sigma, x) = y^{-1} \cdot y'$
    - Check whether $(x\ltimes, y'') \in S$
    - If so, calculate add $\texttt{cont}_f(\ltimes, x) = y^{-1} \cdot y''$
  - Remove $t$ from $H$ if there are any mismatches
  - Add $t$ to $K$ otherwise

# Previous approach

- The process is inefficient because we are, again, effectively performing nested reads of the sample
- Let $l$ be the number of training instances
  Let $m$ be the length of the longest input string
  Let $n$ be the length of the longest output string
  - There can be up to $lm$ pairs in $Z$ (the "snapshot" of $f^p$)
  - The initial search takes up to $m$ steps per pair
  - We run an additional search through the "found" pairs, of which there can be up to $lm$
  - The additional search takes up to $n$ steps per pair
  - Overall we have $\mathcal{O}(lm(m + lmn))$
- As above, the desired result can be obtained without nested reads by using a Prefix Tree Transducer (PTT)

# Current approach

- We do not need to re-calculate contributions directly from the sample every time
- Instead, we can cycle through the transitions $(q, \sigma, u, q')$ in the onward PTT:
  - Check whether $q$ is supported (i.e., $\exists (q, r) \in Z$)
  - If yes, check whether $q'$ is supported (i.e., $\exists (q', r') \in Z$)
  - If yes, put the transition in a bin tied to the input element $\sigma$ and the tier element $\texttt{suff}^1_H(q)$
  - If any bins contain mismatches, the corresponding tier element gets removed from the tier hypothesis

# Current approach

- Let $l$ be the number of training instances
  Let $m$ be the length of the longest output string
  Let $n$ be the summed length of the input strings
  - There are $l + n$ transitions in the onward PTT
  - Checking whether a state is supported takes at most $n$ steps
  - Getting the tier suffix of a state label takes at most $m$ steps
  - Overall we have $\mathcal{O}((l + n)(m + n))$
  - While $\Sigma$ and $\Delta$ are technically constants for the learning problem, the runtime is linear in $|\Sigma|$ and quadratic in $|\Delta|$
- Using a PTT reduces time complexity from $n^5$ to $n^2$

# SL versus subsequential

- Using ISL and OSL functions instead of subsequential functions sacrifices expressiveness, but comes with a gain in learning efficiency.
- Where the runtime of OSTIA is cubic, the runtimes of the ISLFIA and OSLFIA are both quadratic
- The time improvement of the ISLFIA over OSTIA is analogous to our elimination of nested reading
  - OSTIA can undo state merges whereas the ISLFIA cannot

# Strictly local tier-based functions

- For tier-based functions, the same time-expressiveness trade-off previously applied only when the tier was known
- Burness and McMullin (2019) showed that the tier of a $\text{TSL}_2$ function could be learned in quintic time
  - We have shown that this can be reduced to quadratic time
- We have focused on the $\text{OTSL}_2$ class throughout; only slight modifications are needed for $\text{ITSL}_2$ functions
- Other slight modifications can learn particular classes of multi-tiered functions (Burness and McMullin, 2020)

# Other advantages

- Subsequential computation is sufficiently expressive for
  - Vowel harmony (Heinz and Lai, 2013)
  - Consonant harmony (Luo, 2017)
  - Consonant dissimilation (Payne, 2017)
- TSL functions, however, provide more intuitive analyses (Andersson et al., 2020; Burness et al., 2021)
- TSL functions exclude un-phonological behaviours like modulo counting and minimum distance requirements (Burness et al., 2021)

# Future directions

- Tier learning for functions with $k > 2$
  - The tier of tier-based *languages* can be learned for any $k$ (Jardine and McMullin, 2017; Lambert, 2021)
- Online learning instead of batch learning
  - Tier-based languages can be learned in batch (Jardine and McMullin, 2017) or online (Lambert, 2021)
- Tier learning for partial functions
  - Strategy assumes a total function
  - Domain and range information allow OSTIA to learn partial functions (Oncina and Varó, 1996; Castellanos et al., 1998)

# Acknowledgements

This research was supported by the Social Sciences and Humanities Research Council of Canada

Social Sciences and Humanities Research Council of Canada

Conseil de recherches en sciences humaines du Canada

Canada

# References I

Andersson, S., Dolatian, H., and Hao, Y. (2020). Computing vowel harmony: The generative capacity of search and copy. In Baek, H., Takahashi, C., and Hong-Lun Yeung, A., editors, *Proceedings of the 2019 Annual Meeting on Phonology*.

Burness, P. and McMullin, K. (2019). Efficient learning of Output Tier-Based Strictly 2-Local functions. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 78–90. Association for Computational Linguistics.

Burness, P. and McMullin, K. (2020). Multi-tiered strictly local functions. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 245–255. Association for Computational Linguistics.

# References II

Burness, P., McMullin, K., and Chandlee, J. (2021). Long-distance phonological processes as tier-based strictly local functions. *Glossa*, 6.

Castellanos, A., Vidal, E., Varó, M. A., and Oncina, J. (1998). Language understanding and subsequential transducer learning. *Computer Speech and Language*, 12:193–228.

Chandlee, J., Eyraud, R., and Heinz, J. (2014). Learning Strictly Local subsequential functions. *Transactions of the Association for Computational Linguistics*, 2:491–503.

Chandlee, J., Eyraud, R., and Heinz, J. (2015). Output Strictly Local functions. In *Proceedings of the 14th Meeting on the Mathematics of Language (MOL 2015)*, pages 112–125.

# References III

de la Higuera, C. (2010). *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York.

Heinz, J. and Lai, R. (2013). Vowel harmony and subsequentiality. In Kornai, A. and Kuhlmann, M., editors, *Proceedings of the 13th Meeting on the Mathematics of Language (MOL 13)*, pages 52–63, Sofia, Bulgaria. Association for Computational Linguistics.

Heinz, J., Rawal, C., and Tanner, H. G. (2011). Tier-based strictly local constraints for phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 58–64, Portland, OR. Association for Computational Linguistics.

# References IV

Jardine, A. and McMullin, K. (2017). Efficient learning of Tier-Based Strictly k-Local languages. In *International Conference on Language and Automata Theory and Applications (LATA 2017)*, pages 64–76.

Lambert, D. (2021). Grammar interpretations and learning TSL online. In Chandlee, J., Eyraud, R., Heinz, J., Jardine, A., and van Zaanen, M., editors, *Proceedings of the Fifteenth International Conference on Grammatical Inference*, volume 153 of *Proceedings of Machine Learning Research*, pages 81–91. PMLR.

Luo, H. (2017). Long-distance consonant agreement and subsequentiality. *Glossa: A Journal of General Linguistics*, 2:1–25.

# References V

Oncina, J., Garcia, P., and Vidal, E. (1993). Learning subsequential transducers for pattern recognition tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:448–458.

Oncina, J. and Varó, M. A. (1996). Using domain information during the learning of a subsequential transducer. In Miclet, L. and de la Higuera, C., editors, *Grammatical Interference: Learning Syntax from Sentences*, number 1147 in Lecture Notes in Artificial Intelligence, pages 301–312. Springer, Berlin.

Payne, A. (2017). All dissimilation is computationally subsequential. *Language*, 93:353–371.