Output Languages of Strictly Local Phonological Processes

Anton Hampe
Department of Linguistics
Yale University
anton.hampe@yale.edu

Abstract

This paper investigates the relationship between strictly local phonological processes and strictly local phonotactic constraints. On the theoretical side, I identify phonological rewrite rules that do not produce strictly local output languages and that do not weakly preserve the class of strictly local languages. Empirically, I find that strictly local rules without strictly local output languages are largely absent from the PBase database.

1 Introduction

This work refines the computational typology of phonological rewrite rules. I assume basic familiarity with the mechanics of phonological rewrite rules and elementary automata theory. I recommend Hayes (2011) for readers unfamiliar with phonological rewrite rules and Hopcroft et al. (2001) for readers unfamiliar with automata and transducers. Phonological rewrite rules have a long history in linguistics, going as far back as the grammars of Pāṇini from the 5th century BCE (Kiparsky, 2022). Today, they are often referred to as SPE-style rules, following Chomsky and Halle (1968). SPE-style rules stand in a special relationship to the regular languages, which are the string sets recognized by finite state acceptors, and the rational relations, which are the string to string maps computable by finite state transducers. The nature and history of that relationship is summarized below. Johnson (1972) noted that both the schema for morpheme structure used at the time and the environments targeted by phonological rewrite rules specify regular sets of strings, concluding that phonological surface patterns have to belong to the regular class as well. Kaplan and Kay (1994) made this result more rigorous and showed that all SPE-style rules and their common modes of application produce rational relations and regular surface languages. The optimality-theoretic grammar formalisms that are commonly used today generally do not share this property (Lamont, 2021a,b). However, typological work studying the extensions of phonological processes and surface languages suggests that these entities themselves have these properties, independent of the grammar formalisms used to model them (Heinz, 2018).

(Mielke, 2008) compiled PBase, a large database of rewrite rules used in phonological grammars. The existence of such a large database makes refining the computational properties of classes of rewrite rules particularly useful for typological work.

2 Strictly local languages

As computational phonology has matured, the early results suggesting that phonological surface patterns are regular have been refined to more restrictive, subregular classes (Heinz, 2018). One linguistically very useful class of subregular languages is the class of strictly local (SL) languages introduced by McNaughton and Papert (1971). Edlefsen et al. (2008) found that about 75% of stress patterns cataloged in the stresstyp database (Goedemans et al., 2015) belonged to the SL class. The class of SL languages is very robust, in the sense that it permits multiple equivalent definitions in terms of structural restrictions on finite-state automata, logic, closure properties, and algebraic properties (McNaughton, 1974; Rogers et al., 2012; Rogers and Pullum, 2011; Lambert, 2025).

For the purposes of this paper the characterizations in terms of automata and the characterization in terms of logical constraints are the most relevant. Each strictly local language has a canonical acceptor, such that each state is uniquely labeled by a suffix of length $n \geq 0$ of all paths leading to it. If such an acceptor exists, then the language is $(n+1)$-SL. For instance, a machine with a single state labeled by the empty string is 1-SL. Figure 1 shows

the canonical automaton recognizing the 2-SL language $a\{a,b,c\}^*b \cup c\{a,b,c\}^*b$. This language contains all strings over the alphabet $\{a,b,c\}$ that end with b, but do not begin with b. The $\rtimes$ and $\ltimes$ symbols are special markers for the left and right boundary of the string, respectively. These markers are assumed to always mark the boundaries of the string, but are not included in the alphabet of string-internal symbols. In the context of phonology, the symbol # often plays a similar role in marking both the left and right boundaries of a word.

Alternatively, a strictly local language can be characterized in terms of a finite set of allowed or prohibited substrings, also known as n-grams. The language of the automaton in Figure 1 can be characterized as all possible strings containing only length-two substrings or bigrams from the set $\{\rtimes a, \rtimes c, aa, ab, ac, ba, bb, bc, ca, cb, cc, b\ltimes\}$. Alternatively, it can be characterized as the set of all strings over the alphabet $\{a,b,c\}$ not containing the substrings or n-grams $\{\rtimes\ltimes, \rtimes b, a\ltimes, c\ltimes\}$. In the context of phonology, an n-gram like ab being absent from a language is often phrased as the language having the constraint $*ab$, or satisfying the constraint $*ab$.

3 Strictly local functions

Just as string sets can be classified into subregular classes, string transductions can be classified into subrational classes. Subclasses of the rational relations are often named for subregular classes of languages that stand in a special relationship to them. For this paper the input strictly local (ISL) and output strictly local (OSL) classes introduced by Chandlee (2014) are of particular relevance. These classes are named for the fact that they can be structurally defined using canonical transducers that closely mirror the structure of the canonical acceptors underlying SL languages. Canonical ISL transducers have each state uniquely labeled with a suffix of length $n \geq 0$ of the input projection of all paths leading to it. Similarly, canonical OSL transducers have each state uniquely labeled by a suffix of length $n \geq 0$ of the output projection of all paths leading to it. Both classes of transductions have connections to natural classes of phonological rules. ISL functions include phonological rewrite rules $A \rightarrow B / C__D$ under simultaneous application, where A and B consist of a single letter or the empty string and C and D represent strings of bounded length. OSL functions include phonological rewrite rules $A \rightarrow B / C__D$ under iterative application, where A and B consist of a single letter or the empty string and C and D represent strings of bounded length. Processes which can be represented in both ways are included in both classes.

4 Output languages

Belonging to the ISL or OSL classes is not the only way in which a transduction may relate or share properties with the SL languages. The output language of a transduction is the set of strings the transduction produces when fed arbitrary input. In other words, it is the image of Σ^*, where Σ is the input alphabet for the transduction. Given the prevalence of SL phonotactic surface patterns, one might expect the ISL and OSL phonological rules enforcing these surface patterns to have SL output languages. For instance, final devoicing is a phonological process active in Turkish, Russian, German and many other languages. It is commonly modeled by a rewrite rule of the form $\{+voice, -sonorant\} \rightarrow \{-voice\} / __\#$. This rewrite rule produces a strictly local output language with the singular constraint $*\{+voice, -sonorant\}\#$, which makes sense since the fact that such strings are missing from the surface phonotactics of these languages is precisely what motivated the rule in the first place.

Chandlee et al. (2014) show that this is not a trivial property. Belonging to the ISL class, OSL class, or both is not enough to ensure that the output language of a transduction is strictly local. However, so far no theoretical work had been conducted checking whether simplex ISL or OSL processes which can be represented as single rewrite rules have SL output languages.

5 SL-preservation

A class of transductions weakly preserves a class of languages if the transduction produces an image that belongs to the class, for each set of inputs that belong to the class (Hashiguchi and Honda, 1976). For example, a function is weakly SL-preserving if it maps each SL subset of its domain to a SL image. Since the free monoid Σ^* forms a strictly local language, being weakly SL-preserving entails having a SL output language.

I am not aware of any decision procedure for weak SL-preservation, and to my knowledge, the work of Chandlee et al. (2014) is the only work that touches on this property. The question of whether

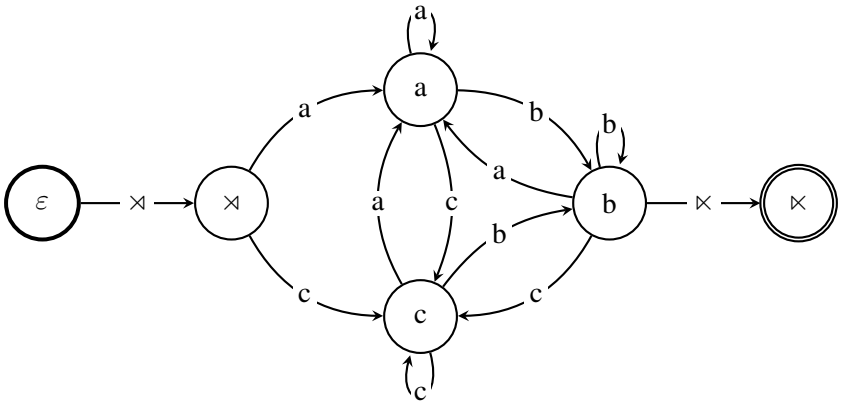

Figure 1: The canonical 2-SL automaton recognizing $a\{a,b,c\}^*b \cup c\{a,b,c\}^*b$. The initial state is marked with a thick outline, and the final state is marked with a double outline.

any ISL or OSL functions representable as simplex phonological rules do (not) weakly preserve SL languages has not been previously explored.

This work offers no decision procedure for SL-preservation either. Instead, I introduce a method which produces counterexamples to SL-preservation for a large number of SPE-style rules. The counterexamples involve creating a strictly local language which consists of alternations of contexts in which the rule applies and surface strings which result from such an application. For example, the rule $A \rightarrow B / C__$ maps the language $(CACB)^+$ to $(CBCB)^+$, regardless of whether it is applied iteratively left-to-right, right-to-left, or simultaneously. The input language is 3-SL and generated by the positive grammar $\{\rtimes CA, CAC, ACB, CBC, BCA, CB\ltimes\}$. The output language is equivalent to $(CB)^{2n}$ and thus not star-free.

This construction is not universal. It fails for rules such as $A \rightarrow B / A__$. Here, the language $(AAAB)^+$ is mapped to $(ABBB)^+$ when the rule is applied iteratively right-to-left or simultaneously. Since the image is a SL language, this is not a valid counterexample. This can be remedied by introducing a distinct symbol that separates the two strings, as in $(AACABC)^+$. The image of that language is then $(ABCABC)^*$, which is not star-free. This construction works, as long as there is some symbol C in the input alphabet which is not implicated in the rule in any way. As I have shown, a substantial subset of SPE-style rules does not weakly preserve the SL class. This means that the property of having a SL output language is not necessarily preserved under composition. Let f and g be two SPE-rules that produce strictly local output languages, but which do not weakly preserve the full SL class. The function f may map the set of all possible input strings to a strictly local language L different from Σ^*. Because g does not weakly preserve the SL class, we have no guarantee that it maps L to a SL language. The composition $g \circ f$ would then map Σ^* to a non-SL language. I return to this issue in section 6.4.

6 Theoretical results

6.1 Methodology

Hulden (2009) introduced the FOMA software package for creating finite-state machines in the machine-readable .att format. I generated .att files for all rewrite rules over the alphabet $\Sigma = \{A, B, C, D, E, F\}$ with context length of up to 2 on either side of the target. Target and change of each rule had to be distinct from each other and were restricted to be either the empty string or a single letter. The inclusion of the empty string allows for epenthesis and deletion rules. The rule involving the largest number of distinct symbols was thus $A \rightarrow B / CD__EF$. For each rule, I created automata under simultaneous application (guaranteeing membership in the ISL class) and left-to-right iterated application (guaranteeing membership in the OSL class). The resulting .att files were analyzed using the LTK software package of Lambert (2024). For this paper, I treat rules that differ only in symbol names as identical. For example, the rules $A \rightarrow B / CC__$, and $A \rightarrow B / DD__$ are treated as the same rule.

$\emptyset \rightarrow x / x__x$	$\emptyset \rightarrow x / x__xx$	$\emptyset \rightarrow x / x__$	$\emptyset \rightarrow x / yy__$
$\emptyset \rightarrow x / xy__$	$\emptyset \rightarrow x / xx__x$	$\emptyset \rightarrow x / xx__xx$	$\emptyset \rightarrow x / xx__$
$\emptyset \rightarrow x / __x$	$\emptyset \rightarrow x / __yy$	$\emptyset \rightarrow x / __yx$	$\emptyset \rightarrow x / __xx$

Figure 2: Rules producing non-SL output languages under simultaneous application.

6.2 Simultaneous application

I found that under simultaneous application there are indeed simplex SPE-style rules that produce non-SL output languages. The set of such rules is shown in Figure 2.

Notably, all such rules were epenthesis rules. I would also like to note that many of these rules are intuitively strange or phonologically atypical. Consonant epenthesis tends to be trigerred by adjacent vowels, and vowel epenthesis tends to be trigerred by illicit consonant clusters (Blevins, 2008; Hall, 2011). However, most of the rules in Figure 2 appear to copy an element from their environment. A third interesting point is that all these rules not only produce output languages which are non-SL, they produce output languages which are non-star free. The rule $\emptyset \rightarrow x / x__$ for instance produces an output language whose intersection with x^* is x^{2n}, i.e. strings of xs with an even length. This is because it maps each string x^n to the string x^{2n}. Since SL languages are closed under intersection, this makes the output language non-SL.

6.3 Iterative application

I found that there are also rules which produce non-SL output languages under iterated left-to-right application. This set of rules is given in Figure 3.

Notably, this set is a subset of the rules with non-SL output languages under simultaneous application. The rules that produce SL output languages under iterated left-to-right application, but not under simultaneous application are rendered with strike-through. When opting for right-to-left application, the rules that lose the property of non-SL output languages are the mirror images $\emptyset \rightarrow x / __yy$ and $\emptyset \rightarrow x / __yx$.

6.4 Composition

Because of the combinatorics of composing every rule with a SL output language with every other rule with a SL output languages, composition of these rules was not tested exhaustively. When randomly sampling rules and composing them, I found that the property of having a SL output language is not always preserved under composition.

The instances of two composed rules forming a complex process with a non-SL output language had a predictable relationship to the simplex rules with non-SL output languages. All of them included an initial step of epenthesis followed by a step of erasing distinctions between triggers and targets. For instance, applying the rule $\emptyset \rightarrow A / B__$ maps strings of the form B^n to strings of the form $(BA)^n$. Following this application with the rule $A \rightarrow B / B__$ in turn maps strings of the form $(BA)^n$ to strings of the form $(BB)^n$. Because the output language of the first rule lacks strings of the form B^+, and the second conspires with the first to add all strings of the form B^{2n}, the overall effect is a non-star-free output language. Neither rule on it's own produces a non-SL output language.

7 Empirical results

7.1 Methodology

I searched PBase for rules matching the template in Figure 2. Because I used concrete individual symbols in my analysis of phonological rules, but PBase (and phonologists generally) use natural classes of segments, I decided a rule matched my template if it fit my template for any subset of the alphabet. For example, a rule $\emptyset \rightarrow j / \{i, j\}__$ would count as matching the template $\emptyset \rightarrow x / x__$, even though $\{i\}$ and $\{i, j\}$ are different sets.

7.2 Potential matches

The search yielded nine matches from seven languages.[1] Eight turned out to be faulty data entry, meaning the prose description of the rule did not match the way the rule was coded. For instance, the database included a rule for glide insertion in Chrau of the form:

$\emptyset \rightarrow$ j/{g, c, b, l, d, h, k, j, m, ʃ, n, p, ɲ, t, w, ʔd, r, ʔb, ŋ, ɟ, ʔ}{c, ɟ, ʃ, ɟ, j}__

[1]The IDs of the matching rules are 595, 2000, 2320, 2349, 2558, 3030, 3225, 6261, and 6669 from the languages Agarabi, Burmese, Chrau, Czech, Eastern Ojibwa, Kiowa, and Sie.

$\emptyset \rightarrow x/x__x$	$\emptyset \rightarrow x/x__xx$	$\emptyset \rightarrow x/x__$	~~$\emptyset \rightarrow x/yy__$~~
~~$\emptyset \rightarrow x/xy__$~~	$\emptyset \rightarrow x/xx__x$	$\emptyset \rightarrow x/xx__xx$	$\emptyset \rightarrow x/xx__$
$\emptyset \rightarrow x/__x$	$\emptyset \rightarrow x/__yy$	$\emptyset \rightarrow x/__yx$	$\emptyset \rightarrow x/__xx$

Figure 3: Rules producing non-SL output languages under left-to-right iterated application. Rules rendered with strike-through produce non-SL output languages under simultaneous application, but not under left-to-right application.

However, the original grammar by Thomas (1971) contains the following description of the rule:

"When a palatal (ch, j, nh, y, s) is in C_f position, there is usually a phonetic y-glide between the vowel and the consonant."

This description implies that a vowel should be included in the right context. Including such a vowel prevents the rule from matching the templates $\emptyset \rightarrow x/xx__$ and $\emptyset \rightarrow x/xy__$.

The ninth match in the database is the rule from Kiowa below.

$\emptyset \rightarrow$ j/{j}{h,ʔ}__

The original grammar of Watkins (1980) describes the rule as follows.

"The palatal glide spreads across the laryngeals yielding a glide onset, a brief moment of coarticulation and a glide release. The rule operates across word boundaries as well as word internally."

This description does not force the glide to be followed by a vowel. The rule as described matches the template $\emptyset \rightarrow x/xy__$. This rule template produces a non-SL output language under simultaneous application, but not under left-to-right or right-to-left application.

Let us take a closer look at how the rule $\emptyset \rightarrow x/xy__$ produces a non-SL output language under simultaneous application. The crucial surface forms are strings of the form $(xy)^n$. The rule template will only produce such strings for even n. Take the output string $xyxy$ and let us underline all the segments that must have been present in the input. Since the rule does not create any new y segments, all the instances of y must have been underlying in the input, giving us $\underline{xy}\underline{xy}$. Since the first x could not have been preceded by the correct context for epenthesis, it too must have been present in the input. This leaves us with $\underline{xy}\underline{xy}$. The last remaining x has the context for insertion underlyingly adjacent. Therefore, the last x must have been epenthesized by the rule.

Now if we append another xy to that output string, we get $\underline{xy}\underline{xy}\underline{xy}$. The y we appended must have been present in the input for the same reason as the other ys. Namely that the phonological rule in question cannot epenthesize a y. The x must have been present in the input, since it is not preceded by the context that would trigger epenthesis.

However, this poses a dilemma. We have established that the final xy in the output must have been underlying. But if it were underlying, it should have triggered the rule and there should be another final x epenthesized at the end. This contradiction shows that $xyxyxy$ cannot be in the output language of the rule. The same argument holds for all $(xy)^n$ for odd n. Patterns such as this one that involve modulo counting fall outside the class of strictly local languages.

In terms of the Kiowa rule, this means one would need repeated instances of jhʔ or jʔh in underlying forms to get similar behavior. The grammar does not contain such underlying forms featuring repeated iterations of laryngeals and glides. This means that the rule could potentially be modified to include another type of segment, such as a vowel, in its context. Given these possibilities for reanalysis and the fact that the rule produces a SL output language when applied left-to-right (as might be expected of a spreading process that proceeds to the right), PBase supports the conclusion that strictly local phonological processes produce strictly local output languages to a striking degree.

8 Conclusion

This work identified the set of simplex phonological rules belonging to the ISL and OSL classes with context length up to 2 which produce non-SL output languages. All such rules were epenthesis rules with highly unusual contexts. A search of the PBase database produced no rules that produce non-SL output languages under iterated left-to-right ap-

plication. The database does contain a rule that would produce a non-SL output language under simultaneous application, if taken at face value. This paper also showed that the property of weakly preserving SL languages is not one that holds for a large majority of simplex ISL and OSL rewrite rules, including very common rule templates.

This second fact makes the property of producing SL output languages unstable under composition. One rule can produce an output language for which a following rule does not preserve strict locality. Understanding under which conditions rules that individually produce SL output languages may compose to yield a complex process which does not is a possible direction for future research. Empirical work investigating whether such combinations of rules exist in analyses of natural phonological systems is another possible direction.

Interestingly, all ISL and OSL process without SL output languages identified in this paper had output languages outside the star-free class. It stands to reason that ordered rewrite rule analyses enforcing counting constraints would have been noticed by now, since the notion that phonology does not count has long been a core assumption of phonology.

If it is the case that the dichotomy between SL output languages and non-star-free output languages is not an accident, that opens up interesting possibilities for restrictions on the ordering of simple phonological rules that might be of use for the study of opacity and rule interaction.

If it is the case that morpheme structure constraints exist, the theoretical portion of this paper implies restrictions on such constraints. Even if constraints on underlying representations were merely SL, a bad combination of morpheme structure constraints and simple phonological processes could conspire to produce non-star-free surface languages.

A final possible application of these results is the design of learning algorithms. Although these results eliminate relatively few simplex processes, they do add a new typologically supported bias that a learner could exploit.

References

Juliette Blevins. 2008. Consonant epenthesis: Natural and unnatural histories. *Language universals and language change*, pages 79–107.

Jane Chandlee. 2014. *Strictly local phonological processes*. University of Delaware.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2014. Learning strictly local subsequential functions. *Transactions of the Association for Computational Linguistics*, 2:491–504.

Noam Chomsky and Morris Halle. 1968. *The Sound Pattern of English*. Harper & Row, New York.

Matt Edlefsen, Dylan Leeman, Nathan Myers, Nathaniel Smith, Molly Visscher, and David Wellcome. 2008. Deciding strictly local (sl) languages. In *Proceedings of the midstates conference for undergraduate research in computer science and mathematics*, volume 6, pages 6–75. Citeseer.

Rob Goedemans, Jeffrey Heinz, and Harry van der Hulst. 2015. StressTyp2, version 1. Web download archive. Available at http://st2.ullet.net.

Nancy Hall. 2011. Vowel epenthesis. *The Blackwell companion to phonology*, pages 1–21.

Kosaburo Hashiguchi and Namio Honda. 1976. Properties of code events and homomorphisms over regular events. *Journal of Computer and System Sciences*, 12(3):352–367.

Bruce P Hayes. 2011. *Introductory phonology*. John Wiley & Sons.

Jeffrey Heinz. 2018. The computational nature of phonological generalizations. *Phonological typology, phonetics and phonology*, 23:126–195.

John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. 2001. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65.

Mans Hulden. 2009. Foma: a finite-state compiler and library. In *Proceedings of the Demonstrations Session at EACL 2009*, pages 29–32, Athens, Greece. Association for Computational Linguistics.

C. Douglas Johnson. 1972. *Formal Aspects of Phonological Description*. Number 3 in Monographs on Linguistic Analysis. Mouton, The Hague.

Ronald M Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational linguistics*, 20(3):331–378.

Paul Kiparsky. 2022. Pāṇini. In *The Oxford History of Phonology*. Oxford University Press.

Dakotah Lambert. 2024. System description: A theorem-prover for subregular systems: The language toolkit and its interpreter, plebby. In *Functional and Logic Programming*, pages 311–328, Singapore. Springer Nature Singapore.

Dakotah Lambert. 2025. On the structure of sets testable in the strict sense. In *Proceedings of the 16th Meeting on the Mathematics of Language (MOL 2025)*. To appear.

Andrew Lamont. 2021a. Optimality theory implements complex functions with simple constraints. *Phonology*, 38(4):729–740.

Andrew Lamont. 2021b. Optimizing over subsequences generates context-sensitive languages. *Transactions of the Association for Computational Linguistics*, 9:528–537.

Robert McNaughton. 1974. Algebraic decision procedures for local testability. *Mathematical systems theory*, 8(1):60–76.

Robert McNaughton and Seymour A Papert. 1971. *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press.

Jeff Mielke. 2008. The emergence of distinctive features.

James Rogers, Jeffrey Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. 2012. Cognitive and sub-regular complexity. In *International Conference on Formal Grammar*, pages 90–108. Springer.

James Rogers and Geoffrey K Pullum. 2011. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, 20(3):329–342.

David D Thomas. 1971. Chrau grammar. *Oceanic Linguistics Special Publications*, (7):i–258.

Laurel Jayne Watkins. 1980. *A grammar of Kiowa*. University of Kansas.