Algebraic Classification of Reduplicative Processes

Matthew Hayden
Stony Brook University
matthew.hayden@stonybrook.edu

Abstract

This paper offers an updated perspective on the computational complexity of reduplication. Since one-way deterministic transducers cannot model reduplication in a straightforward way, the phenomenon has long been considered the outlier of morphology from a complexity perspective. Drawing on algebraic methods, I show that the vast majority of reduplicative processes belong to a few remarkably simple classes of subregular functions. A detailed study of the RedTyp database (Dolatian and Heinz, 2019) reveals that 100% of the surveyed reduplicative processes correspond to string-to-string functions in the class $\mathbb{D}\mathbf{A}$, while over 98% are locally testable ($\mathbb{L}\mathbf{J}_1$) and over 87% are locally trivial ($\mathbb{L}\mathbf{1}$). These results indicate a new upper bound on the complexity of reduplication that is comparable to that of morphological processes in general.

1 Introduction

The starting point for much work in computational morphology is to regard morphological processes as string-to-string functions. Morphological investigation from the perspective of *subregular linguistics* tries to identify the simplest subclasses of string-to-string functions that account for the range of attested processes. It is well-known that one-way deterministic finite-state transducers (1DFTs) are sufficient to capture almost all attested morphological transformations, with total reduplication ($w \rightarrow ww$) being the exception (Roark and Sproat, 2007). In fact, the vast majority of phonological and morphological transformations can be modeled with input strictly local (ISL) or output strictly local (OSL) functions, which are restricted subclasses of the functions computable by a 1DFT (Chandlee, 2014). Again, Chandlee (2014) notes that total reduplication is the exception. Given that most everything else in morphology appears to be subregular, it seems odd that total reduplication is uniquely challenging to compute, despite occuring in around 85% of languages in the World Atlas of Language Structures (WALS) (Rubino, 2013). This apparent imbalance of complexity has remained a puzzle in subregular morphology. Prior research has explored what changes can be made to the conventional finite-state machinery in order to correctly model reduplication (Dolatian and Heinz, 2020, cf. Wang, 2021). This paper pursues an alternative approach that uses tools from the algebraic theory of semigroups to identify universal properties of attested reduplicative functions.

Reduplication is a typologically pervasive phenomenon wherein some amount of phonetic material is copied to induce a change in meaning. Consider the following examples, from Indonesian (Cohn, 1989) and Samoan (Moravcsik, 1978) respectively:

(1) buku → bukubuku
'Book' → 'Books'

(2) alofa → alolofa
'He loves' → 'They love'

While total reduplication, as in (1), is most common, 75% of languages in WALS additionally have partial reduplication, as in (2). In partial (or bounded) reduplication, a subcomponent of the input known as the *target* is copied to create the *reduplicant*. For instance, the target in (2) is the syllable *lo* in *alofa*, while the reduplicant is the second *lo* in *alolofa*. In cases of total (full, or unbounded) reduplication, the target of the reduplication is the entire input form. While the bounded nature of partial reduplication renders it computable by 1DFT (Chandlee, 2014), one-way transducers are incapable of computing total reduplication ($w \rightarrow ww$), because they make only one pass over the input.

More recently, Dolatian and Heinz (2020) argue that both total *and* partial reduplication are better represented by two-way deterministic finite-state transducers (2DFTs) which are strictly more

Proceedings of the Society for Computation in Linguistics (SCiL) 2026, pages 447–459

powerful than their one-way counterparts. These machines are able to compute functions in the class of concatenated OSL functions (C-OSL), which intuitively correspond to the transformations computable by multiple OSL functions with concatenated outputs. The class of C-OSL functions is shown to be sufficiently expressive for the majority of attested reduplicative patterns. Thus, C-OSL offers a tighter characterization of the space of morphological processes than does the full range of functions computable by 2DFTs (i.e. the *regular* functions).

However, Lambert and Heinz (2024) recently leveraged algebraic methods to reclassify phonological processes typically regarded as OSL (or related output-oriented classes) with respect to the properties satisified by their *syntactic semigroups*. Given a 1DFT that models a concrete phonological transformation, one can mechanically construct its syntactic semigroup, which serves as an algebraic description of the effect different alphabet symbols have on the states of the machine. The resulting semigroup structure can be probed with the algebraic decision procedures documented by Lambert (2022). Lambert and Heinz (2024) used these techniques to show that those OSL transductions actually have semigroups in relatively weak subregular classes. This paper extends that line of investigation to the two-way transducers largely classified as C-OSL by Dolatian and Heinz (2020).

The first contribution of this paper is an algorithmic explanation of how to construct transition semigroups for two-way machines, toward the end of facilitating future research in this vein. While the knowledge of how to do this has been implicit in the literature at least as far back as Birget (1989), I provide explicit instructions for how to infer the basis of the transition semigroup of an arbitrary two-way machine. The second contribution is a direct extension of the efforts of Dolatian and Heinz (2020) in the classification of reduplication, by way of a comprehensive study of the RedTyp database (Dolatian and Heinz, 2019). The third contribution comes in the form of a Python script for computing two-way semigroups automatically, and for the classification of the transducers in RedTyp.

Section 2 gives an overview of and motivation for two-way models of reduplication (Dolatian and Heinz, 2019, 2020). In Section 3, I show how to classify reduplicative processes with respect to the algebraic properties satisfied by their finite-state implementations. In Section 4, I apply these methods to the finite-state transducers catalogued in RedTyp, a database of reduplicative patterns across 91 languages (Dolatian and Heinz, 2019). Ultimately, the results suggest that reduplication is not as computationally complex as it is thought to be—the majority of reduplicative processes fall into three or more rather simple subclasses of the algebraic subregular hierarchy.

2 Background

One of the coarsest classifications of a transformation is whether it can be modeled by a one-way deterministic finite-state transducer (1DFT), a machine that computes a left-to-right (or right-to-left) function from an input string to an output string. The symbols in the input come from a finite alphabet Σ, and the symbols in the output string come from a finite alphabet Ω. In this paper, all the transducers happen to have identical input and output alphabets. I use the abbreviation $\Sigma_{\rtimes}$ when Σ is enriched with two delimiters $\rtimes$ and $\ltimes$ that mark the left and right edges of a word.

Definition 2.1 (1DFT)**.** A one-way deterministic finite-state transducer is a four-tuple $\langle Q, q_i, F, \Delta \rangle$, where Q is a set of states, $q_i \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and Δ is a set of transition instructions. The instructions in Δ are of the form $(p, i, q, o) \in Q \times \Sigma_{\rtimes} \times Q \times \Omega^*$ such that Δ contains at most one instruction corresponding to each $(p, i) \in Q \times \Sigma_{\rtimes}$.

While it is well-known that 1DFTs cannot compute total reduplication, Dolatian and Heinz (2020) argue that 1DFT models also fail to capture partial reduplication in a linguistically plausible way. This argument appeals to the *origin semantics* (Bojańczyk, 2014), which formalize the correspondence between the input and output symbols in a transduction.

As an example, consider the leftmost diagram Figure 1, which describes the one-way computation of $\rtimes$CVC$\ltimes$ $\mapsto$ CVCVC. The lines indicate the correspondences between the input and the output. A one-way machine that computes this function starts on the left edge symbol $\rtimes$, then moves left-to-right through the string, writing symbols to the output. Once the machine sees the first V, it outputs VCV, so as to emulate initial-CV reduplication. This diverges from linguistic intuitions because the C in the reduplicant is associated with a V in the base. In this respect, one-way models fall short of correctly capturing the dependencies linguists

Figure 1: Origin semantics of Initial CV Reduplication from a 1DFT (left) and a 2DFT (right)

are trying to model. Therefore, they argue that reduplication necessitates the jump to two-way deterministic finite-state transducers (2DFTs), which can compute a more complex class of functions. 2DFTs achieve this greater expressive power by removing the restriction that input strings be processed strictly left-to-right or right-to-left, so that the machine may move back and forth on the input during the course of computation.

Definition 2.2 (2DFT)**.** A two-way deterministic finite-state transducer is a four-tuple $\langle Q, q_i, F, \Delta \rangle$, where Q is a set of states, $q_i \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and Δ is a set of transition instructions. The instructions in Δ are of the form $(p, i, q, o, d) \in (Q \times \Sigma_{\rtimes} \times Q \times \Omega^* \times D)$ such that Δ contains at most one instruction corresponding to each $(p, i) \in Q \times \Sigma_{\rtimes}$, and where where $D = \{1, -1\}$. When an instruction has $d = 1$, computation proceeds by moving one symbol to the *right* in the input string. When $d = -1$, computation proceeds by moving one symbol to the *left* on the input string.

The diagram on the right in Figure 1 demonstrates a two-way transducer's origin semantics on the same computation. Whereas the one-way machine is forced to move in one direction, the two-way machine is allowed to revisit the first C in the input before writing the reduplicant. Thus, both the Cs and Vs in the reduplicant are associated correctly with their correspondents in the base. So, with respect to the origin semantics, 2DFTs are better suited for computing partial reduplication than 1DFTs. Given that 2DFTs are also capable of computing total reduplication, it seems that 2DFTs are a better model for reduplicative processes in general.

In fact, Dolatian and Heinz (2020) show that virtually all reduplicative processes are modelable with 2DFTs, and offer a computational typology of reduplication by identifying weaker subclasses of 2DFTs that maintain strong empirical coverage. Notably, approximately 87% of RedTyp transducers were shown to be C-OSL functions. However, while OSL functions (and related classes) have often been invoked in the analysis of non-local phonological transformations, algebraic analysis reveals that these seemingly output-oriented functions fall into more constrained classes when classified only with respect to their inputs. Lambert and Heinz (2024) take a second look at which are phonological processes in the literature that originally motivated OSL, namely iterative spreading (Chandlee et al., 2015), and (a)symmetric harmony (Heinz, 2010). While these long-distance processes were previously understood as OSL (or tier-based OSL) patterns, their semigroup structures reveal them to be in the subregular algebraic class $\mathbb{L}\mathbf{1}$, or its tier-based extension. Known also as the *locally trivial* or *generalized definite* class, $\mathbb{L}\mathbf{1}$ constraints also naturally accomodate local phonotactics (Lambert, 2026), and so membership in $\mathbb{L}\mathbf{1}$ is taken to be a reasonable level of complexity for a phonological process or pattern.

That these OSL functions have such restricted subregular semigroups is of particular interest because, as proven by Lambert (2022), the class OSL resists an algebraic characterization. It turns out that *any* finite semigroup is the syntactic semigroup of some OSL transducer, meaning the examined output-oriented processes in principle could have had algebraic structures of arbitrary regular complexity. Given that the current understanding of reduplication relies on C-OSL, it stands to wonder whether reduplicative transducers also have surprisingly simple semigroups, potentially shifting our estimation of reduplicative complexity in line with the rest of morphophonology. To test this hypothesis, I apply algebraic methods to a database of reduplicative transducers, generating and classifying semigroups for each with a Python script, ultimately yielding a robust typology of reduplicative complexity.

3 Methods

When it comes to reduplication, 2DFT models seem prefable to 1DFT models, both because they can compute total reduplication and because they yield the appropriate origin semantics in cases of partial reduplication. While reduplicative 2DFTs are known to be mostly C-OSL (Dolatian and

Heinz, 2020), the algebraic approach may uncover a computationally simpler generalization. This section describes how to investigate this hypothesis, with concrete results from a database study. Section 3.1 describes the database itself, Section 3.2 details how to construct semigroups for two-way transducers, and Section 3.3 provides classification procedures for the subregular classes which dissect the space of attested reduplications.

3.1 Database: RedTyp

RedTyp (Dolatian and Heinz, 2019) is a typological database of reduplication that comes furnished with finite-state models of 138 reduplicative processes. These processes span 91 languages, ultimately consisting of 57 distinct 2DFTs. The data in RedTyp was drawn largely from surveys (Moravcsik, 1978; Rubino, 2005; Inkelas and Downing, 2015) with the goal of collecting diverse processes. While fairly comprehensive, it should be noted that RedTyp obviously does not contain an exhaustive list of all documented reduplicative transformations. Even still, its empirical coverage and supply of finite-state models make RedTyp a database conducive to computational study.

Although RedTyp contains 138 examples of reduplication according to Dolatian and Heinz (2019), I was only able to extract the data of 136 processes via SQL query. Three of those processes were missing the field corresponding to the 2DFT, leaving 133 transducers available for study. A text file containing all 133 2DFTs, as well as the code for computing and classifying their semigroups, is available on Github.[1]

3.2 Semigroups for two-way transducers

Each example of reduplication catalogued in RedTyp is associated with a 2DFT that can be classified algebraically in a procedural way. The current work on algebraic classification (see: Lambert, 2026 and references therein) extracts syntactic semigroups from one-way machines, which can then be mechanically checked for set membership across many subregular classes. However, there are a notions of semigroups for 2DFTs as well. I adopt the definition of *transition semigroup*[2] for two-way automata mentioned originally by (Birget, 1989) and explicated more recently by (Carton and Dartois, 2015; Kunc and Okhotin, 2011). The extraction of the transition semigroup from a 2DFT is also mechanical, and the subregular decision procedures apply to all semigroups in the same way, regardless of whether they were obtained from one-way or two-way machines. Therefore, it is possible to automate the classification of RedTyp transducers to get an informative view of the reduplicative complexity landscape in one fell swoop. In this section, I detail how to construct these transition semigroups step-by-step.

As a running example of semigroup construction, consider this pattern of initial consonant reduplication, found in Agta (Moravcsik, 1978):

(3) liw → lliw
'scold' → 'angry'

The first step in the classification of a morphological processes is to settle on an analysis and a 2DFT. The transducer given in Figure 2, which models this process in Agta, is adapted only slightly from its instantiation in RedTyp (for details, see the discussion in Limitations).

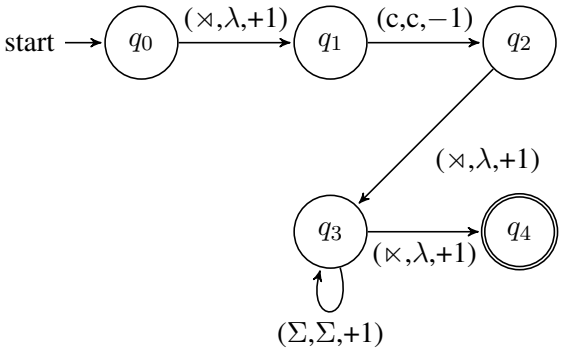

Figure 2: 2DFT for Agta Initial-C Reduplication

When computing the transition semigroup for a one-way machine, one partitions Σ^+ into the equivalence classes of input words that have the same effect on the states of the machine. The question is: Starting from a state p, when an input string is processed left-to-right (or right-to-left), in which state q does the machine end up? In a two-way machine, however, there are more ways that an input string can be processed besides just left-to-right or right-to-left. Because the machine can move back and forth on the input, it is possible to start on the left edge of the input string and exit on the left edge. Likewise, it is possible to start on the right edge and exit the string on the left. This leaves four possibilities: a string can be processed left-to-right,

[1] www.github.com/mphayden1/classifying-reduplication

[2] The syntactic semigroup of a 1DFT corresponds to the transition semigroup of its *canonical* form; for 2DFTs, no canonical form is known (see Limitations).

left-to-left, right-to-right, and right-to-left. These are called the *behaviors* of a string.

Definition 3.1 (Behaviors)**.** Let $\mathcal{A} = \{Q, q_i, F, \Delta\}$ be a two-way automaton, and let $w \in \Sigma^+$. Then the left-to-left behaviors of w are given by:
$b_{ll}(w) = \{(p, q)$: There exists a run over $\mathcal{A}$ such that $\mathcal{A}$ starts on the leftmost symbol of w in state p and leaves w from the left edge in state q.$\}$
The other three behaviors, $b_{lr}, b_{rr},$ and b_{rl} are defined analogously. Let $B(w) = \langle b_{ll}, b_{lr}, b_{rl}, b_{rr} \rangle$.

Notice that for a given single symbol $a \in \Sigma$, it is necessarily the case that $b_{ll}(a) = b_{rl}(a)$, and that $b_{lr}(a) = b_{rr}(a)$. This is because the leftmost and rightmost symbol are the same for a unigram.

Definition 3.2 (Transition Semigroup)**.** A semigroup is a set that is closed under an associative binary operation. Let $\mathcal{A} = \{Q, q_i, F, \Delta\}$ be a two-way automaton. Let $\sim_B$ be the equivalence relation such that for $u, v \in \Sigma^+$, $u \sim_B v$ iff $B(u) = B(v)$. Then the *transition semigroup* of $\mathcal{A}$ is given by the quotient $\Sigma^+ / \sim_B$.

For a two-way machine $\mathcal{A}$, despite a possibly infinite input language consisting of words $w \in \Sigma^+$, there is a finite set of behaviors $S = \{B(w_1), B(w_2), \ldots, B(w_k)\}$ such that the behavior of every word in Σ^+ has a behavior in S. That set, when equipped with a particular product operation $\bullet$, forms the transition semigroup $\langle S, \bullet \rangle$. The $\bullet$ operation was originally defined by Birget (1989), and it describes how to faithfully compose the behaviors $B(u), B(v)$ of two words in order to recover the behavior of their concatentation $B(uv)$.

Theorem 3.1 (Concatenation of inputs)**.** Let $\mathcal{A} = \{Q, q_i, F, \Delta\}$ be a two-way automaton. Let u and v be words over Σ^+, with behaviors $B(u)$ and $B(v)$. Let $\bullet$ be the operation defined as:

$$
\begin{aligned}
&B(u) \bullet B(v) = \\
&\langle b_{ll}(u) \cup b_{lr}(u)[b_{ll}(v)b_{rr}(u)]^* b_{ll}(v) b_{rl}(u), \\
&\ b_{lr}(u)[b_{ll}(v)b_{rr}(u)]^* b_{lr}(v), \\
&\ b_{rl}(v)[b_{rr}(u)b_{ll}(v)]^* b_{rl}(u), \\
&\ b_{rr}(v) \cup b_{rl}(v)[b_{rr}(u)b_{ll}(v)]^* b_{rr}(u) b_{lr}(v) \rangle
\end{aligned}
$$

The juxtaposition of behaviors $b_{xy}(w)b_{x'y'}(w')$ denotes their composition as relations. The star * is the reflexive-transitive closure of the relation enclosed in brackets. Then, the following formula holds:

$$B(u) \bullet B(v) = B(uv)$$

The key fact is that the behaviors of a word uv can be inferred directly from the behaviors of u and v, which entails that the entire transition semigroup is generated from the behaviors of the single-letter words in Σ. These could be thought of as the primitive behaviors, which form the basis of the semigroup. Once the primitive behaviors are known, one can discover the rest of the semigroup elements through their repeated $\bullet$-composition, until no new behaviors are discovered.

The second step, then, is to identify these primitive behaviors. To identify the primitive behaviors of a two-way machine, consider one symbol at a time, and consider its effect on the machine starting at each state. Refer back to the Agta transducer $\mathcal{A}$ in Figure 2. This 2DFT operates over alphabet symbols c and v, which means that we only need to directly calculate $B(c)$ and $B(v)$, and the rest of the semigroup elements will follow. Recalling that $b_{ll}(c) = b_{rl}(c)$ and $b_{lr}(c) = b_{rr}(c)$, it suffices to check only the left-left and left-right behaviors of c. In Figure 2, the only states that have outgoing arrows labeled with c are q_1 and q_3. Because the c-transition from q_1 to q_2 has a -1 as its direction, it moves the head of the transducer to the left of the C in the input. Thus, $\mathcal{A}$ leaves starts in q_1 on the left edge of C, and leaves c on the left edge in state q_2. Since this is the only place in the machine where this action occurs, we conclude that $b_{ll}(c) = b_{rl}(c) = (q_1, q_2)$. The only other c-transition in $\mathcal{A}$ occurs as the self-loop on q_3. It has a positive direction component, meaning $\mathcal{A}$ starts on the left edge of c in state q_3 and exits c on the right in state q_3, so $b_{lr}(c) = b_{rr}(c) = (q_3, q_3)$. Performing the same procedure for v yields the following two behaviors, for $B(c)$ and $B(v)$:

$$\langle \{(q_1, q_2)\}, \{(q_3, q_3)\}, \{(q_1, q_2)\}, \{(q_3, q_3)\} \rangle$$
$$\langle \emptyset, \{(q_1, q_2), (q_3, q_3)\}, \emptyset, \{(q_1, q_2), (q_3, q_3)\} \rangle$$

The third step is to $\bullet$-compose each of the primitive behaviors with each other, and then with the products of those compositions, and so on, until no new behaviors are found. Following the formulas given in 3.1, we can compute $B(cv)$ directly from the behaviors of c and v. This yields a distinct behavior:

$$B(cv) = \langle \{(q_1, q_2)\}, \{(q_3, q_3)\}, \emptyset, \{(q_3, q_3)\} \rangle$$

While writing in-line, I use the notation [w] to refer to the semigroup element corresponding to $B(w)$. Because $B(c)$, $B(v)$, and $B(cv)$ are all

distinct, that yields at least three semigroup elements: [c], [v], and their product [cv]. In fact, these three elements are all and only those in $S = \Sigma^+/\sim_B$. One can check this by verifying [v] • [c] = [v] • [v] = [v], and so on until every pairwise combination has been tried. Every •-composition of two of these elements in S evaluates to another in S, so S is closed under •-composition. The structure of the semigroup $\langle S, \bullet \rangle$ is summarized in Table 1.

•	c	v	cv
c	cv	cv	cv
v	v	v	v
cv	cv	cv	cv

Table 1: Multiplication table for Agta

To some extent, these can be reasoned about with common sense. In S, [v] corresponds to all words starting with a vowel, which are rejected. So, concatenating anything on the end of a word with behavior [v] should evaluate to another word with behavior [v], since the new word would still start with a vowel. The only string with behavior [c] is c itself, and any word that starts with c and is longer than one symbol has behavior [cv]. Thus, concatenating anything to the end of these words will yield a word with behavior [cv].

3.3 Classifying two-way semigroups

The final step is the classification of the semigroup, in accordance with the decision procedures collected by Lambert (2022). I detail only some of these procedures here, which suffice to test membership in the subregular classes that turn out to be relevant for reduplication. Figure 3 provides an inclusion chart for the algebraic classes at hand, where **1** corresponds to a point in the intersection of **D** and **K**.

As discussed previously, much work has gone into finding subclassifications of 1DFTs that provide a tight fit for the types of phonological processes that are attested. For example, ISL and OSL functions, and their tier-based extensions (Chandlee et al., 2014). A function f is k-ISL if there is some finite k such that the output of the function at any step in the computation is decided by the last $k-1$ symbols in the *input*. We say f is ISL if it is k-ISL for any k. Analogously, f is k-OSL if the output at any point in computation is decided by the last $k-1$ symbols in the *output*. Together, these classes offer impressive empirical coverage (for details, see Chandlee, 2014; §7.1). While the ISL class covers local phonological processes, the ability for OSL functions to track output symbols allows them to model long-distance processes, like iterative spreading.

Coming from another angle, the algebraic subregular hierarchy presented in (Lambert, 2022) provides a fine-grained taxonomy of string-to-string functions, plus a suite of algorithmic decision procedures for the classification of transducers. These two perspectives are connected: Lambert and Heinz (2023) showed that the class of total ISL functions is exactly the algebraic class **D** of the *definite* functions. On the other hand, however, the OSL class has no algebraic correspondent. In fact, for any finite semigroup S, there exists an OSL function which has S as a semigroup (Lambert, 2022). This motivated Lambert and Heinz (2024) to reexamine many prototypical examples of (tier-based) OSL functions from through the lens of algebra. Each of the processes, when classified according to semigroups, were shown to belong to the classes **D**, **K**, $\mathbb{L}$**1**, or their tier-based extensions. These classes are near the absolute bottom of the subregular hierarchy—only slightly above the class **1** of *trivial* semigroups. My investigation of the semigroup structure of reduplicative 2DFTs reveals that reduplication likewise has a rather simple computational core.

3.3.1 Idempotence and algebraic identities

The property of idempotence is an important notion in the algebraic study of semigroups, and it plays a critical role in probing subregular complexity. Let S be some semigroup, and $a \in S$. Then a is called *idempotent* if and only $aa = a$. An *aperiodic* semigroup has the special property that all elements a eventually yield an idempotent under repeated composition, meaning there is some k such that $a^n = a^{n+1}$ for all $n \geq k$. For a given a in an aperiodic semigroup, we write a^ω to refer to the unique idempotent obtained under the repeated squaring of a. Then a semigroup S is aperiodic if and only if the identity $a^\omega = a^{\omega+1}$ holds for all $a \in S$. A famous result by Schützenberger (1965) holds that the languages with aperiodic syntactic monoids are precisely those belonging to the subregular class of star-free.

It turns out that every reduplicative transducer in RedTyp is in fact aperiodic. For example, by inspecting Table 1, it is clear that [v] and [cv] are idempotent elements. And while [c] is not an

idempotent, it has an idempotent power, because $[c]^3 = [c] \bullet [c] \bullet [c] = [cv] \bullet [c] = [cv]$. However, this is not especially surprising; after all, star-free is a rather robust class, subsuming most of the subregular classes of linguistic interest. However, there are more restricted subregular classes that correspond to more constraining algebraic identities. The following theorem collects identities from Lambert (2022) which pick out the relevant classes for reduplication. These classes are relatively weak, indicating that reduplicative processes actually cluster near the bottom of the algebraic subregular hierarchy.

Theorem 3.2. The following hold:

- **1** A semigroup S is *trivial* iff it satisfies $x = y$ for all $x, y \in S$.
- **D** A semigroup S is *definite* iff $xa^\omega = a^\omega$ for all $x, a \in S$.
- **K** A semigroup S is *reverse definite* iff $a^\omega x = a^\omega$ for all $x, a \in S$.
- $\mathbb{L}$**1** A semigroup S is *generalized definite* iff $a^\omega x a^\omega = a^\omega$ for all $a, x \in S$.
- $\mathbb{L}\mathbf{J}_1$ A semigroup S is *locally testable* iff $a^\omega x a^\omega y a^\omega = a^\omega y a^\omega x a^\omega$ for all $a, x, y \in S$.
- $\mathbb{D}\mathbf{A}$ A semigroup S is in $\mathbb{D}\mathbf{A}$ iff $(xyz)^\omega y (xyz)^\omega = (xyz)^\omega$ holds for all $x, y \in S$.

Determining which of these identities hold in a semigroup is a matter of straightforward symbol manipulation. For Agta, notice that $[v] \bullet [c] = [v] \bullet [v] = [v] \bullet [cv] = [v]$, and $[cv] \bullet [c] = [cv] \bullet [v] = [cv] \bullet [cv] = [cv]$. This process therefore satisfies the identity $a^\omega x = a^\omega$ for all $x, a \in S$, and so belongs to the class **K** of *reverse definite functions*. By inclusion, this process is also locally trivial, locally testable, and $\mathbb{D}\mathbf{A}$. These facts can be verified by testing the other identities in Theorem 3.2. In the appendix, results for the RedTyp transducers across these six classes are collected in tables.

4 Results

The algebraic analysis of the transducers in RedTyp reveals three subregular classes within which the overwhelming majority of reduplication is contained. Of these classes, the least restricted is $\mathbb{D}\mathbf{A}$, which is the smallest class that contains all 133 transducers. Out of those 133 transducers, 131 were classified as $\mathbb{L}\mathbf{J}_1$, and so $\approx 98\%$ of RedTyp is contained in the intersection of $\mathbb{D}\mathbf{A}$ and $\mathbb{L}\mathbf{J}_1$, which are incomparable (as shown in Figure 3). 116 of the

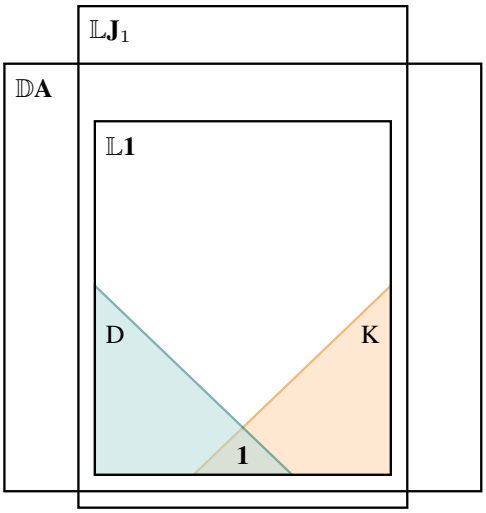

Figure 3: Algebraic subregular hierarchy for reduplication

transducers fall into $\mathbb{L}\mathbf{1}$, which is contained in $\mathbb{L}\mathbf{J}_1$, and so $\approx 87\%$ are also in $\mathbb{D}\mathbf{A} \cap \mathbb{L}\mathbf{1}$. All cases of unbounded reduplication are trivial (**1**), which is the simplest possible type of string-to-string function from the algebraic perspective (a fact first noted by Lambert, 2022).

The classes with the broadest coverage—$\mathbb{D}\mathbf{A}$, $\mathbb{L}\mathbf{J}_1$, $\mathbb{L}\mathbf{1}$—are all of theoretical interest. The class $\mathbb{D}\mathbf{A}$ is exactly the class $\mathrm{FO}^2_<$—that is, the class of functions definable in the fragment of first order logic restricted to two variables, and with the precedence relation $<$ (Thérien and Wilke, 1998). Moreover, these are the class of string functions defined by "partially-ordered" 2DFTs (Schwentick et al., 2001). Intuitively, a partially-ordered 2DFT is a two-way transducer satisfying the constraint that all loops in the machine are self-loops.

Regarding $\mathbb{L}\mathbf{J}_1$, their membership in this class establishes a connection with the locally testable languages (McNaughton, 1974), which are definable in the fragment of propositional logic over substrings (Rogers and Lambert, 2019). Since we can think of $\mathbb{L}\mathbf{J}_1$ transducers as a straightforward generalization of $\mathbb{L}\mathbf{J}_1$ acceptors, local testability of a *function* implies that there is some number k such that the output of the function at any step in computation is decided by the set of seen k-substrings and the last $k-1$ symbols seen. Recalling that local processes are generally computable by ISL_k functions, in which the output is decided only by the last $k-1$ symbols, it seems that the extra power needed to reduplication can be attained by also tracking the seen k-substrings in memory.

The generalized definite class, $\mathbb{L}\mathbf{1}$, is a class of mounting importance in computational mor-

phophonology. $\mathbb{L}\mathbf{1}$, perhaps with tiers, seems to capture a wide range of phonotactic patterns while being sufficienty constrained to be learnable (Lambert, 2026). Again, these results are concerned with sets of patterns, not functions, but an $\mathbb{L}\mathbf{1}$ acceptor becomes an $\mathbb{L}\mathbf{1}$ transducer when outputs are added to the transitions, and those two objects are treated the same under the algebraic approach. Lambert and Heinz (2024) use this to argue that the motivating cases for OSL are better thought of as as (tier-based) $\mathbb{L}\mathbf{1}$. This paper demonstrates the same for over 87% of reduplicative transducers, which indicates that, in broad strokes, reduplication is computationally aligned with morphophonology as a whole.

Discussion

The results are easier to interpret when taking a look at reduplicative processes that distinguish between the different classes. The weakest class in the subregular algebraic hiearchy is **1**, which is the class of functions with trivial semigroups consisting of exactly one idempotent element. Intuitively, this means that the machine treats every string over the alphabet in the same way—it makes no distinction between different segments at any point in the computation. There are 40 processes in RedTyp that belong to this extremely restricted class, and they are are precisely the transducers for total reduplication and triplication. Moving up in complexity, a vast majority of processes have semigroups in $\mathbb{L}\mathbf{1}$, which establishes a computational parallel between reduplication and phonology in general. In fact, while Lambert (2026) argues that the tier-based extension of $\mathbb{L}\mathbf{1}$ is a tight approximation of phonotactic complexity, the reduplicative transducers in $\mathbb{L}\mathbf{1}$ need not reference tiers at all. More interesting, then, are the functions which are at least locally testable ($\mathbb{L}\mathbf{J}_1$). An example comes from Chamorro, which has a reduplicative process that targets the CV in the stressed syllable (Inkelas and Downing, 2015).

(4) hu.gán.do → hu.gá.gan.do
'play' → 'playing'

It makes sense that this process should require more expressive power than those in $\mathbb{L}\mathbf{1}$. $\mathbb{L}\mathbf{1}$ string functions are sensitive to the beginning and end of the input string, but the process in (4) requires a sensitivity to primary stress, which could be arbitrarily deep in the input word.

Out of 133 processes studied, only two had semigroups outside of $\mathbb{L}\mathbf{J}_1$. One of these is heavy CVC reduplication in Ilocano, which can mark plurality (Raimy, 2000):

(5) a. kaldíŋ → kal-kaldíŋ
'goat' → 'goats'

b. paʔíd → paa-paʔíd
'fan' → 'fans'

In (5a), the onset and vowel of the first syllable are copied, along with the next consonant. However, if the proceeding consonant is a glottal stop, as in (5b), the vowel is lengthened instead. Why this subtlety corresponds to a measurable increase in complexity is not clear. The corresponding transducer in RedTyp is complex, with 14 states, and it is possible that an alternative analysis or implementation may reduce its apparent complexity. The other process outside of $\mathbb{L}\mathbf{J}_1$ comes from Malay, and appears to be an instance of back-copying, where the reduplicant undergoes a phonological change that is also reflected in the base. Under the analysis of McCarthy and Prince (1995), in (6) the nasality spreads progressively from the nasal across the reduplicant, and then transfers back to the first vowel by correspondence.

(6) /hamə/ → [hãmə̃ hãmə̃], (*[hamə̃ hãmə̃])
'germ' → 'germs'

This particular example is also discussed by Dolatian and Heinz (2020), who also identify this process as unusually complex compared to others. They show that (6) cannot be computed by a C-OSL function, or even by the concatenation of two unrestricted (not necessarily OSL) 1DFTs. While there may be alternative simplifying analyses or two-way machines that have weaker semigroups, I leave this puzzle for future investigation.

The results of this study suggest that $\mathbb{D}\mathbf{A}$ is a hard upper bound on the complexity of reduplication (though likely not the least upper bound). As mentioned above, functions in $\mathbb{D}\mathbf{A}$ correspond to transformations definable in $\mathrm{FO}^2_<$—so what logically possible reduplications are predicted to be unattested? Let $\mathrm{FO}_<$ be the class of all first-order definable string transductions under the general precedence relation $<$. This class, defined in logic, actually coincides with the class of aperiodic transductions (Carton and Dartois, 2015). $\mathrm{FO}^2_<$ is contained in $\mathrm{FO}_<$, so it follows that reduplication is aperiodic, a fact we corroborate algebraically in Section 3.3.1. Under this view, we do not expect to find periodic (non-$\mathrm{FO}_<$) reduplication. This would

be like having a rule that says, "if *w* is of even length, output *ww*; otherwise, reduplicate only the first CV."

However, the bound of $\text{FO}^2_<$ also rules out the $\text{FO}_<$ functions that require more than two variables to define. One dividing line between $\text{FO}^2_<$ and $\text{FO}_<$ is that the idea of *betweenness* is inexpressible without access to a third variable (Krebs et al., 2020). This means that it is impossible to define regular properties like "an *a* can only precede another *a* if there is a *b* between them" in $\text{FO}^2_<$. We should not expect, then, to see a generalization like "total reduplication, but only if the stem has a vowel between every consonant." Whether or not that prediction bears out universally (though the results suggest it may)—this serves as a nice example of how the algebra makes predictions that are formally precise and eminently testable.

Future directions

One goal of this paper is to attest that algebraic classification is flexible, and can be a fruitful perspective, so long as the linguistic representation at hand is amenable to an algebraic analysis. While I only discussed one- and two-way machines in this paper, there are many equivalent representations for regular string functions that could also be explored. For instance, streaming string transducers are an equivalent model to two-way transducers, that also have a well-defined transition semigroup (Carton and Dartois, 2015). Or, one could forgo the machine entirely, and adopt a representation that uses *characteristic functions* instead (Bojańczyk, 2014), or one that visualizes transductions as languages of *origin graphs* (Bojańczyk et al., 2017). The theory of characteristic functions in particular could potentially be used to strengthen the results in this paper, as a characteristic function serves as the canonical device for a transduction with origin information. Looking beyound string-based representations, future research in this direction could consider trees and tree algebras, which could reveal algebraic insights in the domain of syntax or prosodic stucture.

5 Conclusion

This paper intersects two recent lines of inquiry in computational morphophonology: the development of linguistically plausible finite state models of reduplicative processes (Dolatian and Heinz, 2020) and the classification of phonological transformations using algebraic techniques (Lambert, 2022). I demonstrated how to calculate the syntactic semigroup of a 2DFT, provided a script for doing so automatically, and completed a study of RedTyp, which samples reduplicative processes from 91 languages. Ultimately, it yielded a typology of reduplicative functions and their distribution over some of the simplest subregular classes. That the vast majority of said functions ended up in $\mathbb{D}\mathbf{A} \cap \mathbb{L}\mathbf{J}_1$ identifies a new lowest upper bound on the complexity of reduplication. While the results of the algebraic classification are formal, they are of empirical interest because they describe constraints on the space of possible reduplicative processes that were previously unknown.

Importantly, the results of this study contrast with the traditional perspective on the computational complexity of reduplication. Because one-way machines fail to compute reduplication, it seems as if there is a large jump in computational power needed. In reality, this is not the case—the ability to move back and forth on the input is necessary to capture the correct origin semantics, but reduplicative processes seem not to make use of the extra power it affords. A key observation is that 1DFTs must compute *order-preserving* transductions (Bojańczyk, 2014), which cannot generate crossing associations in the origin semantics (as in Figure 1). This seems to be the property that sets reduplicative maps apart, and motivates the switch to a two-way machine. So, setting order-preservation aside, reduplication seems computationally unexceptional with respect to other morphological processes, spanning classes from $\mathbf{1}$ to $\mathbb{L}\mathbf{J}_1$ and $\mathbb{D}\mathbf{A}$. This upper bound is comparable with that of other complex phenomena outside of reduplication, e.g. ATR vowel harmony in Tutrugbu (Lambert, 2022; §5.6). Given also that RedTyp is majority generalized definite ($\mathbb{L}\mathbf{1}$), reduplication arguably seems run-of-the-mill in computational morphology and phonology. Not only does the algebraic perspective shed new light on the nature of reduplication, then, but also proves its worth as a sophisticated diagnostic for complexity that avoids conflating subregularity with order-preservation.

Acknowledgements

I am grateful to Jeffrey Heinz, Thomas Graf, and Mark Aronoff for their feedback and encouragement. I also thank the three anonymous reviewers for their helpful and detailed suggestions, as well

as the audience of the 2025 Rutgers Subregular Phonology Workshop for entertaining an early presentation of this work.

References

Jean-Camille Birget. 1989. Concatenation of inputs in a two-way automaton. *Theoretical Computer Science*, 63(2):141–156.

Mikołaj Bojańczyk. 2014. Transducers with origin information. In *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II 41*, pages 26–37. Springer.

Mikołaj Bojańczyk, Laure Daviaud, Bruno Guillon, and Vincent Penelle. 2017. Which classes of origin graphs are generated by transducers? In *ICALP 2017*.

Olivier Carton and Luc Dartois. 2015. Aperiodic two-way transducers and fo-transductions. In *24th EACSL Annual Conference on Computer Science Logic*, volume 41, pages 160–174.

Jane Chandlee. 2014. *Strictly local phonological processes*. Ph.D. thesis, 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.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2015. Output strictly local functions. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, pages 112–125, Chicago, USA.

Abigail C. Cohn. 1989. Stress in indonesian and bracketing paradoxes. *Natural Language & Linguistic Theory*, 7(2):167–216.

Hossep Dolatian and Jeffrey Heinz. 2019. RedTyp: A database of reduplication with computational models. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*, pages 8–18.

Hossep Dolatian and Jeffrey Heinz. 2020. Computing and classifying reduplication with 2-way finite-state transducers. *Journal of Language Modelling*, 8(1):179–250.

J. A. Green. 1951. On the structure of semigroups. *Annals of Mathematics*, 54:163.

Jeffrey Heinz. 2010. String extension learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 897–906, Uppsala, Sweden.

Sharon Inkelas and Laura J Downing. 2015. What is reduplication? typology and analysis part 1/2: The typology of reduplication. *Language and linguistics compass*, 9(12):502–515.

Andreas Krebs, Kamal Lodaya, Paritosh K Pandya, and Howard Straubing. 2020. Two-variable logics with some betweenness relations: Expressiveness, satisfiability and membership. *Logical Methods in Computer Science*, 16.

Michal Kunc and Alexander Okhotin. 2011. Describing periodicity in two-way deterministic finite automata using transformation semigroups. In *International Conference on Developments in Language Theory*, pages 324–336. Springer.

Dakotah Lambert. 2022. *Unifying Classification Schemes for Languages and Processes with Attention to Locality and Relativizations Thereof*. Ph.D. thesis, State University of New York at Stony Brook.

Dakotah Lambert. 2026. Multitier phonotactics with logic and algebra. *Phonology*, 43.

Dakotah Lambert and Jeffrey Heinz. 2023. An algebraic characterization of total input strictly local functions. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2023*, pages 25–34, Amherst, MA.

Dakotah Lambert and Jeffrey Heinz. 2024. Algebraic reanalysis of phonological processes described as output-oriented. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2024*, pages 129–138, Irvine, CA.

John J. McCarthy and Alan Prince. 1995. Faithfulness and reduplicative identity. In Jill N. Beckman, Laura Walsh Dickey, and Suzanne Urbanczyk, editors, *Papers in Optimality Theory*. Graduate Linguistic Student Association, University of Massachusetts, Amherst, MA.

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

Edith A. Moravcsik. 1978. Reduplicative constructions. In Joseph H. Greenberg, Charles A. Ferguson, and Edith A. Moravcsik, editors, *Universals of Human Language. Volume 3: Word Structure*, pages 297–334. Stanford University Press, Stanford, CA.

Anil Nerode. 1958. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544.

Eric Raimy. 2000. *The phonology and morphology of reduplication*. Walter de Gruyter.

Brian Roark and Richard Sproat. 2007. *Computational approaches to morphology and syntax*. OUP Oxford.

James Rogers and Dakotah Lambert. 2019. Some classes of sets of structures definable without quantifiers. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 63–77, Toronto, Canada.

Carl Rubino. 2005. Reduplication: Form, function and distribution. *Studies on reduplication*, 28:11–29.

Carl Rubino. 2013. Reduplication. In Matthew S. Dryer and Martin Haspelmath, editors, *WALS Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Marcel Paul Schützenberger. 1965. On finite monoids having only trivial subgroups. *Information and Control.*, 8(2):190–194.

Thomas Schwentick, Denis Thérien, and Heribert Vollmer. 2001. Partially-ordered two-way automata: A new characterization of DA. In *International Conference on Developments in Language Theory*, pages 239–250. Springer.

Denis Thérien and Thomas Wilke. 1998. Over words, two variables are as powerful as one quantifier alternation. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 234–240.

Yang Wang. 2021. Recognizing reduplicated forms: Finite-state buffered machines. In *Proceedings of the 18th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 177–187.

Appendix: Limitations

The chief formal limitation of this paper is that, to my knowledge, there is not yet any notion of a canonical form of a 2DFT, at least under the standard semantics. In the world of one-way machines, an automaton is called *minimal* if its states correspond exactly to the Nerode equivalence classes over its language (Nerode, 1958). This extends directly to transducers. Although Lambert (2022) takes a minimal 1DFT with the sink state removed (if necessary) to be canonical, there is no canonical two-way transducer. This means that any semigroup analysis of a two-way machine is only sufficient to give an upper bound on complexity—for instance, given that a particular two-way machine is in $\mathbb{D}\mathbf{A}$, we only know that the function computed is *at most* $\mathbb{D}\mathbf{A}$.

While Bojańczyk (2014) proves the existence of a canonical device for transductions with origin information, how to apply this result to the 2DFTs in RedTyp is not straightforward. Without a simple way to automatically minimize the machines in RedTyp, any improvements to the machines were made by hand. It is important to note that this can have an effect on their classification. For example, it was noted in Section 3.2 that the transducer for Initial-C reduplication in Agta given in Figure 2 was adapted slightly from RedTyp. Two changes were made: 1. In RedTyp, the direction of the transitions from q_1 to q_2 is mistakenly positive (only in the transducer code; it is correct in the diagram), and 2. There was a self-loop on q_2 with the label $(\Sigma, \lambda, +1)$ that was removed. While that loop does not change the function (because it is impossible to read any alphabet symbols in q_2), it changes the semigroup. Compare the semigroup structure given in 1 to the semigroup of the transducer with the superfluous loop, given in Table 2.

•	c	v	cv	vc
c	c	cv	cv	c
v	vc	v	v	vc
cv	c	cv	cv	c
vc	vc	v	v	vc

Table 2: Agta semigroup, with extra loop

Although the semigroup in Table 2 is in both $\mathbb{D}\mathbf{A}$ and $\mathbb{L}\mathbf{J}_1$, there is not much else to say about it. However, removing the loop on q_2 yields the semigroup in Table 1, which belongs to the more constrained classes $\mathbf{K}$ and $\mathbb{L}\mathbf{1}$. In addition to adjusting the transducer for Agta, I did the same for the seven other languages which have identical transducers, as well as for the transducer for Chinanteco final-C reduplication (as labeled in the appendix). These were done by visual inspection for extraneous loops, but because we lack canonical 2DFTs, future work may in principle find smaller machines with simpler semigroups for any of these machines.

A Appendix: Tables of Results

This appendix contains tables that summarize the results from the study of RedTyp. This appendix also includes the classes $\mathfrak{L}$-Trivial and $\mathfrak{R}$-Trivial, which are properly contained between $\mathbf{D}$ and $\mathbb{D}\mathbf{A}$ and between $\mathbf{K}$ and $\mathbb{D}\mathbf{A}$, respectively. These are classes related to the $\mathfrak{L}$ and $\mathfrak{R}$ relations defined originally by Green (1951). Interested readers are referred to (Lambert, 2022; §4). Table 3 summarizes those machines that do not have semigroups in $\mathbb{L}\mathbf{1}$. Table 4 contains only those transducers that were edited to remove extraneous loops and lower their complexity. Table 5 contains those processes that were in $\mathbb{D}\mathbf{A}$, $\mathbb{L}\mathbf{J}_1$, $\mathbb{L}\mathbf{1}$, but no smaller classes. Lastly, Table 6 contains all processes that are trivial ($\mathbf{1}$), and thereby belong to all classes discussed in this paper.

#	Language	Morpheme ID	$\mathbb{D}$**A**	$\mathbb{L}\mathbf{J}_1$	$\mathbb{L}$**1**	$\mathfrak{L}$-Triv
1	Armenian	arm-ECHO-Echo	✓	✓		✓
2	English	eng-derisive-Echo	✓	✓		✓
3	Turkish	tur-ECHO-Echo	✓	✓		✓
4	Tuvan	Tuvan-ECHO-Echo	✓	✓		✓
7	Kannada	kan-ECHO-Echo	✓	✓		✓
8	Kolami	Kolami-ECHO-Echo	✓	✓		✓
9	Tamil	tam-ECHO-Echo	✓	✓		✓
58	Bikol	bik-IMPF-InitialCV	✓	✓		
59	Ewe	ewe-CONT/NMNZ?-InitialCV	✓	✓		
60	Tagalog	tgl-unclear-InitialCV	✓	✓		
61	Tohono O'odham	TohonoOodham-PL-InitialCV	✓	✓		
68	Oykangand	Oykangand-unclear-InitialCVC	✓	✓		
89	Mangarayi	Mangarayi-PL/INT-InitialVC	✓	✓		
91	Chamorro	cha-CONT-Stressed	✓	✓		
94	Samoan	smo-PL-PenultimateCV	✓	✓		
74	Ilocano	ilo-PL/PROG-InitialCVC	✓			
98	Malay	may-unclear-Total	✓			

Table 3: Processes that are not $\mathbb{L}$**1**

#	Language	Morpheme ID	$\mathbb{D}$**A**	$\mathbb{L}\mathbf{J}_1$	$\mathbb{L}$**1**	$\mathfrak{L}$-Triv	**D**	$\mathfrak{R}$-Triv	**K**
10	Chinanteco	Chinanteco-PLPoss-FinalC	✓	✓	✓	✓	✓		
35	Agta	Agta-DIM-InitialC	✓	✓	✓			✓	✓
36	Bikol	bkl-polysemous-Initial C	✓	✓	✓			✓	✓
37	Marshallese	mah-ADJZ-InitialC	✓	✓	✓			✓	✓
38	Nupe	Nupe-unclear-Initial C	✓	✓	✓			✓	✓
39	Pacoh	Pacoh-denominal verbalization-Initial C	✓	✓	✓			✓	✓
40	Shilh	Shilh-PROG-InitialC	✓	✓	✓			✓	✓
41	Yakan	Yakan-repeated/distributed-Initial C	✓	✓	✓			✓	✓
42	Yoruba	yor-polysemous-Initial C	✓	✓	✓			✓	✓

Table 4: Processes that had extraneous loops removed

#	Language	Morpheme ID	#	Language	Morpheme ID
77	Agta	Agta-PL-InitialCVC	24	Mokilese	Mokilese-NMNZ-FinalCVC
43	Akan	aka-unclear-InitialCV	67	Mokilese	Mokilese-unclear-InitialCVC
20	Ao Naga	AoNaga-DISTR-FinalCVC	75	Mokilese	Mokilese-PROG-InitialCVC
13	Arrente	Arrente-INSTR-FinalCV	81	Ngiyambaa	Ngiyambaa-unclear-InitialCVCV
44	Attic Greek	grc-PERF-InitialCV	47	Nitinaht	Nitinaht-resemble-InitialCV
62	Aztek	Aztek-xbyx-InitialCV	23	Pacoh	Pacoh-DISTR-FinalCVC
69	Aztek	Aztek-PROG-InitialCVC	64	Papago	Papago-PL-InitialCV
84	Balangao	Balangao-unclear-InitialCVCV	26	Paumari	Paumari-unclear-FinalCVCV
56	Bikol	bkl-unclear3-InitialCV	55	Pima	Pima-PL-Initial CV
11	Chamorro	cha-INT-FinalCV	30	Quileute	Quileute-FREQ/DISTR-FirstCafterFirstV
93	Chuckchee/Koryak	Chuckchee/Koryak-ABS-NonlocalInitialCVC	63	Quileute	Quileute-PL-InitialCV
66	Chumash	Chumash-unclear-InitialCVC	48	Rotuman	Rotuman-CONT,FREQ,HAB?-InitialCV
92	Creek	mus-PL-NonlocalInitialCV	72	Samala/Chumash	Samala/Chumash-REP-InitialCVC
18	Dakota	dak-VRBZ-FinalCVC	33	Semai	Semai-unclearFirstLastC
78	Dyirbal	Dyirbal-moreX-InitialCVCV	27	Siriono	Siriono-CONT-FinalCVCV
85	Dyirbal	Dyirbal-unclear-InitialCVCV	73	Somali	som-PL-InitialCVC
79	Fox	Fox-CONT?-InitialCVCV	76	Southern Paiute	SouthernPaiute-unclear-InitialCVC
14	Hausa	hau-PL-FinalCV	49	Sundanese	sun-CONT-InitialCV
45	Hausa	hau-INT/PLUR-InitialCV	50	Sundanese	sun-pretend-InitialCV
15	Hiaki/Yaqui	Hiaki-HAB-InitialCV	51	Sundanese	sun-transitivize-InitialCV
16	Hopi	Hopi-unclear-FinalCV	25	Tagalog	tgl-DIM-FinalCVCV
70	Ilocano	ilo-IMPF-InitialCVC	52	Tagalog	tgl-NMNZ-InitialCV
19	Kaingang	Kaingang-PL-FinalCVC	53	Tagalog	tgl-unclear2-InitialCV
82	Kinande	Kinande-unclear-InitialCVCV	57	Tagalog	tgl-unclear3-InitialCV
17	Kwaza	Kwaza-PROG?-FinalCV	65	Tagalog	tgl-IMPF-InitialCV
31	Levantine Arabic	ara-INT-FirstCafterSecondC	86	Tagalog	tgl-unclear-InitialCVCV
80	Limos Kalinga	LimosKalinga-ITER	34	Temiar	Temiar-ACT/CONT-FirstLastC
46	Lushootseed	Lushootseed-DIM-InitialCV	28	Thao	Thao-INT-FinalCVCV
71	Lushootseed	Lushootseed-DISTR-InitialCVC	54	Tsimshian	tsi-unclear-InitialCV
88	Lushootseed	Lushootseed-OutofControlEtc-InitialVC	29	Tzeltal	Tzeltal-unclear-FinalVC
83	Makassarese	Makassarese-unclear-InitialCVCV	87	Warlpiri	Warlpiri-unclear-InitialCVCV
12	Manam	Manam-polysemous-FinalFoot	90	Washo	was-PL-InitialVCV
21	Marshallese	mah-VBZ-FinalCVC	32	Zuni	zun-REP-FirstCafterSecondC
22	Mokilese	Mokilese-VRBZ-FinalCVC			

Table 5: Processes that are $\mathbb{D}\mathbf{A}$, $\mathbb{L}\mathbf{J}_1$, and $\mathbb{L}\mathbf{1}$

#	Language	Morpheme ID	#	Language	Morpheme ID
95	Amele	Amele-ITER-Total	112	Pacoh	Pacoh-RECP-Total
96	Boumaa Fijian	BoumaaFijian-INT-Total	131	Shipibo	Shipibo-CONT-Triplication
102	Dholuo	Dholuo-mitigation-Total	113	Somali	som-toXAtAllSides-Total
103	Dyirbal	Dyirbal-PL-Total	114	Sundanese	sun-INT-Total
104	Dyirbal	Dyirbal-RECP-Total	115	Sundanese	sun-notEvenX-Total
129	Emai	Emai-ADVZ-Triplication	116	Sundanese	sun-RECP-Total
105	Ewe	ewe-XbyX-Total	117	Swahili	swa-DIM-Total
5	Hindi	hin-ECHO-Echo	119	Tagalog	tgl-everyX-Total
106	Hungarian	hun-occassionally-Total	118	Tausug	Tausug-unclear-Total
99	Indonesian	ind-INT/REP-Total	132	Telugu	tel-emphasis?-Triplication
100	Indonesian	ind-RECP-Total	120	Thai	tha-DIM-Total
107	Indonesian	ind-PL-Total	121	Thai	tha-INT-Total
108	Kanuri	Kanuri-glossonym-Total	122	Twi	twi-DenomADJZ-Total
6	Kashmiri	kas-ECHO-Echo	133	twi	twi-ADVZ-Triplication
109	Malay	may-similar-Total	123	Tzeltal	tzeltal-CONT-Total
97	Mandarin	chi-INT?-Total	124	Tzeltal	Tzeltal-DISTR-Total
101	Mandarin	chi-every-Total	125	Tzeltal	Tzeltal-veryMuch-Total
110	Mokilese	Mokilese-PROG-Total	126	Yami	Yami-RECP-Total
130	Mokilese	Mokilese-CONT-Triplication	127	Yessan-Mayo	YessanMayo-DISTR-Total
111	Nez Perce	NezPerce-DIM-Total	128	Yoruba	yor-PL-Total

Table 6: Processes that are trivial ($\mathbf{1}$)