Morphologically simplex D-quantifiers are strictly 2-local

Thomas Graf
Department of Linguistics & Institute for Advanced Computational Science
Stony Brook University
mail@thomasgraf.net

Abstract

Even though languages can express a wide range of quantifiers, only a small number are ever realized as morphologically simplex determiners: *every*, *no*, *some*, and *most*. This is puzzling because I) *most* is much more complex than the other three, and II) quantifiers like *an even number* are simpler than *most* yet cannot belong to this class. Building on concepts from subregular complexity, I present a new way of measuring a quantifier's complexity in terms of its verification pattern. The quantifiers *every*, *no*, *some*, and *most* all have strictly 2-local (SL-2) verification patterns, but quantifiers like *an even number* do not. This suggests that subregular complexity, and in particular strict locality, plays a crucial role for how much meaning can be packed into morphologically simplex expressions.

1 Introduction

The literature on generalized quantifiers (see Keenan and Westerståhl 1996, Peters and Westerståhl 2006 and references therein) considers a wide range of quantificational expressions, from *every*, *no*, and *some* to *not all*, *all but one*, *most*, *at most half*, *an even number*, *a third*, *between two and eight*, or *more - than*. It is noteworthy, though, that across languages the majority of these expressions are structurally complex, involving multiple words or morphemes. For instance, there seems to be no language with a single word that has the same meaning as *not all*. This is particularly well-documented in the case of determiners. Among *D-quantifiers*, i.e. quantifiers that function as determiners, the only simplex ones (*modulo* agreement markers) are realizations of *every*, *no*, *some*, and *most*, although not all of them are instantiated in every language.

Surprising as this may be, it becomes even more puzzling once one considers the complexity of these quantifiers. Semantic automata theory (van Benthem, 1986; Steinert-Threlkeld and Icard, 2013) allows us to determine a quantifier's position in the Chomsky-hierarchy of string languages (Chomsky, 1956, 1959; Chomsky and Schützenberger, 1963). Many quantifiers are regular, including simplex *every*, *no*, and *some*, but also the morphologically complex expressions *not all*, *all but one*, and *an even number*. On the other hand, *most* belongs to the more complex class of context-free string languages. If *most* can be a morphologically simplex D-quantifier (MSDQ), why isn't this possible for some quantifiers of lower complexity?

Recently, I set out to refine this picture in Graf (2019b) by drawing from work on the subregular complexity of patterns in phonology, morphology, and syntax (see Chandlee 2017, 2022, Heinz 2018, Dolatian and Heinz 2020, Graf 2022a,b, Hanson 2023a,b, and references therein). I argued that among the regular quantifiers, *every*, *no*, *some*, *not all*, and *all but one* are particularly simple because they belong to the subregular class of tier-based strictly local languages (Heinz et al., 2011; Lambert and Rogers, 2020), whereas *an even number* does not. While this explains how *an even number* differs from these quantifiers, it still does not explain why *not all* and *all but one* cannot be MSDQs, and it actually widens the complexity gap between *most* and the other MSDQs.

In this paper, I propose that the contradictory complexity results are resolved by adopting *verification patterns* as a new string model of quantifier interpretation. A verification pattern for quantifier Q encodes instructions for how the elements of the domain can be arranged to easily determine whether the statement expressed by Q is true. The complexity of Q is equated with the complexity of the simplest possible verification pattern for Q. The MSDQs *every*, *no*, *some*, and *most* all have strictly 2-local verification patterns, but *an even number* does not. Verification patterns thus place the attested MSDQs within the same complexity class while correctly excluding many other quantifiers.

The paper is laid out as follows. After a brief discussion of the notion of *quantifier languages* from semantic automata theory (Sec. 2.1), I define what it means for a string language to be strictly k-local (Sec. 2.2). I then define verification patterns as an alternative to quantifier languages (Sec. 3.1) and show that all MSDQs have SL-2 verification patterns, in particular *most* (Sec. 3.2 and 3.3). But not all quantifiers with SL-2 verification patterns are MSDQs, and Sec. 3.4 proposes several conditions that separate the MSDQs from the other quantifiers in this class. The paper concludes with some tentative observations on how this approach could be extended to handle infinite domains, various pragmatic effects, capture typological frequency effects, and cognitive parallels to syntax (Sec. 4).

2 Background

2.1 Semantic automata and the typology of quantifiers

Generalized quantifiers like *every*, *no*, *some*, and *most* are formally modeled as type $\langle 1, 1\rangle$ quantifiers, i.e. as functions that take two sets A and B as arguments and return a truth value depending on whether a specific relation holds of A and B.

Example 1. The quantifier *every* corresponds to the function $f_{every} : E \times E \to \{0, 1\}$ (where E is some fixed set of entities) such that $f_{every}(A, B) = 1$ iff $A \subseteq B$. In the sentence *every cat sneezed*, A is the set of cats and B is the set of entities that sneezed. The sentence is true iff the set of cats is a subset of the set of sneezers. ⌟

The *semantic automata* approach (van Benthem, 1986; Steinert-Threlkeld and Icard, 2013) makes it possible to recast any type $\langle 1, 1\rangle$ quantifier Q as a string language L_Q over the alphabet $\{0, 1\}$. We also call L_Q a *quantifier language*. Intuitively, one constructs a binary string s^A_B such that each position i of s^A_B corresponds to a distinct element $a_i \in A$, and the symbol at position i is 1 if $a_i \in B$ and 0 otherwise. Given a quantifier Q, then, $f_Q(A, B) = 1$ iff $s^A_B \in L_Q$. Crucially, for all quantifiers discussed in this paper this must hold no matter how elements are associated to positions, so $s^A_B \in L_Q$ iff L_Q contains every permutation of s^A_B.

Example 2. Continuing the previous example, suppose that the discourse salient set A of cats consists of Mocha, Murli, and Cinderella, whereas the set B of sneezers consists of Mocha and Mary. In this scenario, it is false that every cat sneezed, and the semantic automata approach expresses this as follows.

First, Mocha is replaced with 1, whereas Murli and Cinderella are each mapped to 0. With one 1 and two 0s, we can build three binary strings: 100, 010, and 001. The quantifier language of *every* consists of all strings that do not contain 0. For if some element $a \in A$ is replaced by 0, then $a \notin B$ and thus $A \nsubseteq B$. And in the other direction, $A \nsubseteq B$ entails that there is at least one a such that $a \notin B$, and hence s^A_B must contain at least one 0. None of the three binary strings above are members of L_{every}, and thus *every cat sneezed* is correctly predicted to be false for the specific scenario described above. ⌟

With semantic automata, the cognitive complexity of quantifiers can be measured in terms of the complexity of the computational machinery that is needed to generate the corresponding quantifier languages. Tab. 1 lists some well-known complexity results. Note that many quantifier languages actually belong to a proper subclass of the class listed in the table. For example, *most* could be more adequately classified as a deterministic context-free language, or even more tightly as a one-counter language. In Graf (2019b) I showed that *every* and *no* are *strictly 1-local* (SL-1), whereas *some*, *not all*, and *exactly one* are *tier-based strictly 2-local* (TSL-2) — a large reduction in complexity with connections to phonology (McMullin and Hansson, 2015; McMullin, 2016; Jardine and Heinz, 2016; Burness et al., 2021; Mayer, 2021). These refinements do not change the fact, though, that complexity tells us little about what quantifiers may be MSDQs.

Only four MSDQs are attested across languages: *every*, *no*, *some*, and *most* (English *one* does not belong in this category because it is a numeral, and Russian has morphologically simplex *half* but its syntactic behavior is that of a noun rather than a determiner). Why should these be the only MSDQs? Why is it impossible for, say, *an even number of* to ever be realized as an MSDQ? Complexity considerations make this even more puzzling: the class of attested MSDQs contains two that are SL-1, one that is TSL-2, and one that isn't even regular, while excluding many quantifiers of similar or lesser complexity.

Quantifier	Definition	String constraint	Complexity	MSDQ?
every	$A \subseteq B$	no 0	regular	Yes/No
every (existential import)	$A \subseteq B \& A \neq \emptyset$	no 0 and at least one 1	regular	Yes/No
no	$A \cap B = \emptyset$	no 1	regular	Yes
not all	$A \not\subseteq B$	at least one 0	regular	No
some/at least one	$A \cap B \neq \emptyset$	at least one 1	regular	Yes
exactly one	$\lvert A \cap B \rvert = 1$	exactly one 1	regular	No
an even number of	$\lvert A \cap B \rvert \; mod \; 2 = 0$	an even number of 1s	regular	No
half	$\lvert A \cap B \rvert = \lvert A - B \rvert$	an equal number of 1s and 0s	context-free	No
most	$\lvert A \cap B \rvert > \lvert A - B \rvert$	more 1s than 0s	context-free	Yes
less than half	$\lvert A \cap B \rvert < \lvert A - B \rvert$	fewer 1s than 0s	context-free	No
at least one third	$3 \cdot \lvert A \cap B \rvert \geq \lvert A \rvert$	at most three times more 0s than 1s	context-free	No
a prime number of	$\lvert A \cap B \rvert$ is prime	a prime number of 1s	context-sensitive	No

Table 1: A list of common quantifiers with their set-theoretic definition, the string constraint instantiated by their quantifier languages, the complexity of said quantifier language, and whether the quantifier can be expressed as a morphologically simplex determiner

While the goal of Graf (2019b) was to resolve this tension, it actually exacerbates it. On the positive side, the paper showed that *an even number* is more complex than *every*, *no*, *some*, and *not all*, and it observes that among those four, *not all* differs from the three MSDQs with respect to a specific monotonicity property. But the existence of morphologically simplex *most* is still very surprising considering that its quantifier language is not even regular, let alone SL-1 or TSL-2. Following the credo that one person's *modus ponens* is another's *modus tollens*, Graf (2019b) presents this as additional evidence for the proposal by Hackl (2009) that *most* is built up from multiple parts and hence not an MSDQ. But this just begs the question why this option of camouflaging multiple parts as an MSDQ is unavailable for, say, *not all* or *an even number*. The account in (Graf, 2019b) thus fails to reconcile the absence of morphologically simplex *not all* with the existence of morphologically simplex *most*, in particular as the latter has a much more complex quantifier language than the former.

As I will show in Sec. 3, though, the complexity landscape changes greatly if quantifier languages do not need to be closed under permutation. While the complexity of *an even number* remains the same, *most* becomes SL-2 and now is a natural fit for the other three MSDQs. In order to fully appreciate what this means, we have to properly define what it means for a string language to be SL-2.

2.2 Strict locality over strings

Intuitively, a string language is strictly k-local (SL-k; $k \geq 1$) iff it can be described by a finite set of permissible substrings of length k.

Example 3. Consider the string language $L := \rtimes(10)^*\ltimes$, which contains the strings $\rtimes\ltimes$, $\rtimes 10\ltimes$, $\rtimes 1010\ltimes$, $\rtimes 101010\ltimes$, and so on. We can describe L in terms of five permissible bigrams: $\rtimes\ltimes$, $\rtimes 1$, 10, 01, and $0\ltimes$. Every string in L contains only these permissible bigrams (though not necessarily all of them), and every string outside L necessarily contains at least one bigram that is not one of these five permissible bigrams. Since the permissible substrings are of length 2, L is SL-2. ⌟

There is an equivalent characterization of SL-k in terms of forbidden substrings (as long as $k \geq 1$), but the definition with permissible substrings will be easier to use for the purposes of this paper. More specifically, we will define SL-k in terms of positive SL-k grammars.

Given a (finite) alphabet Σ, we use Σ^* to denote the set of all possible strings over Σ, including the empty string ε, and Σ^+ for Σ^* without ε. We furthermore use Σ_E to denote $\Sigma \cup \{\rtimes, \ltimes\}$, where $\rtimes, \ltimes \notin \Sigma$ are left edge and right edge markers, respectively. For any $k \geq 1$, $\Sigma_E^k \subsetneq \Sigma_E^*$ is the set of all strings over Σ_E whose length is exactly k. If Σ contains exactly one symbol σ, then we write σ^k, σ^*, σ^+ instead of $\{\sigma\}^k$, $\{\sigma\}^*$, and $\{\sigma\}^+$, respectively. Given a string s, w is a *k-factor* (or *k-gram*) of s iff $w \in \Sigma_E^k$ and there exist (possibly empty) strings u and v over Σ_E such that $s = uwv$. We write $f_k(s)$ for the set of all k-factors of string s; if the length of s is strictly less than k, then $f_k(s)$ is undefined.

Definition 1 (Strictly k-local). A *(positive) SL-k grammar* over alphabet Σ is a (possibly empty) set $G \subseteq \Sigma_E^k$. The string language generated by G is $L(G) := \{s \mid f_k(\rtimes^{k-1} s \ltimes^{k-1}) \subseteq G\}$. A string language L is SL-k iff there is an SL-k grammar

G such that $L(G) = L$. A string language L is strictly local (SL) iff there is some k such that L is SL-k. ⌟

Every strictly k-local string language is regular as it can be recognized by a deterministic finite-state automaton where each state memorizes the $k-1$ most recent symbols. However, not every regular string language is strictly local.

Example 4. Consider the string language $L := (0^*\ 1\ 0^*\ 1\ 0^*)^*$, which consists of all strings over $\{0, 1\}$ that have an even number of 1s. This includes ε, 11, 1111, 111111, and so on. Suppose this language were strictly k-local for some even k. Then $1^k \in L$ but $1^{k+1} \notin L$. But all the k-factors of $\rtimes^{k-1} 1^{k+1} \ltimes^{k-1}$ (which are $\rtimes^i 1^j$ and $1^j \ltimes^i$ for all $i, j \geq 0$ such that $i + j = k$) are also k-factors of $\rtimes^{k-1} 1^k \ltimes^{k-1}$. With $k = 2$, for instance, $f_2(\rtimes 111 \ltimes) = \{\rtimes 1, 11, 1 \ltimes\} = f_2(\rtimes 11 \ltimes)$. Hence every strictly k-local grammar that generates $1^k \in L$ also generates $1^{k+1} \notin L$, and thus L cannot be strictly k-local. Since k was arbitrary, L is not strictly local. ⌟

In fact, the class SL of strictly local string languages is maximally weak in the sense that no other class has been proposed that includes infinitely many infinite languages and is properly subsumed by SL. The class SL on its own instantiates an infinite hierarchy — the class of SL-k string languages is a proper subclass of the class of SL-$(k + 1)$ languages for all $k \geq 1$. In this paper, I focus on the very bottom of this hierarchy, i.e. SL-1 and SL-2.[1] Since every SL-1 string language is also SL-2, the latter is the more important class for this paper. I argue that all MSDQs are maximally simple in the sense that they have SL-2 *verification patterns*, thus resolving the puzzle posed by *most*.

3 The verificational simplicity of *most*

We are now ready to formulate the central insight of this paper: the complexity of quantifiers can be measured in terms of their verification patterns (Sec. 3.1), and doing so reveals all attested MSDQs to form a natural class in the sense that they are *SL-2 verifiable*, which means that their verification patterns are SL-2 string languages (Sec. 3.2). Admittedly, this hinges on defining *most* as *at least half* instead of *more than half* (Sec. 3.3), and additional restrictions are needed to rule out unattested MSDQs (Sec. 3.4). But this still marks a significant step away from the status of *most* as a complexity outlier among MSDQs.

[1]The class SL-0 can be defined but is pathological. The only possible SL-0 grammar is the empty set $\emptyset$. Depending on whether one interprets $\emptyset$ as a positive grammar or a negative grammar (i.e. a set of forbidden 0-factors), it generates either the empty language or all of Σ^*. This is the only case where the generative capacity of positive and negative SL grammars diverges, which provides good reason not to include SL-0 in the definition of SL.

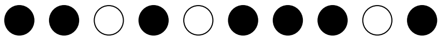
Figure 1: By rearranging the marbles such that there never are two white marbles next to each other, Mary can verify whether most marbles are black without counting all the marbles or calculating their relative proportions.

3.1 From quantifier languages to verification patterns

The complexity results in Tab. 1 hold with respect to quantifier languages that are closed under permutation. The idea behind permutation closure is that the conditions that a quantifier $Q(A, B)$ imposes on A and B hold irrespective of what linear structure one imposes on A. From a linguistic perspective, however, this may distort the cognitive complexity of quantifiers.

Example 5. Suppose the Assistant Dean of the Office of Deranged Tasks has taken a bag with an odd number of marbles in two colors, black and white, and has meticulously arranged them in a line that spans across all the rooms of said office. Mary is then tasked by the Assistant Dean to determine whether most of the marbles are black. Mary cannot eyeball the whole line at once or rely on other heuristics. At first she considers counting, but after a long day of work she does not want to spend the mental effort required to keep track of numbers.

Instead, Mary opts for a simpler solution that does not require counting. She puts all marbles back into the bag and then builds a new line according to the following rules: The first marble must be a black, and each white marble must be immediately to the right of a black marble (see Fig. 1). If Mary ever reaches a point where these rules cannot be met, then it is not the case that most marbles are black. She happily reports her findings to the Assistant Dean, who fires her on the spot for having altered the meticulous marble arrangement. ⌟

While Mary in our example was under an implicit obligation to keep the order of elements undisturbed, this requirement does not hold for the interpretation of quantifiers. The complexity of L_Q

expresses how difficult it is to determine the value of $f_Q(A, B)$ given an arbitrary order of A. An alternative measure would look at how difficult it is to define a verification pattern for f_Q, i.e. a pattern that guarantees that $f_Q(A, B)$ is true iff the elements of A can be arranged according to that pattern.

Definition 2 (Verification pattern). Let L_Q be the (permutation-closed) quantifier language of some type $\langle 1, 1\rangle$ quantifier Q. We call a set V_Q of strings over $\{0, 1\}$ a *verification pattern for* Q iff the permutation closure of V_Q is L_Q. Given a class C of string languages, we say that Q is C *verifiable* iff Q has some verification pattern V_Q in C. ⌟

Note that while quantifier languages are unique, a quantifier may have many distinct verification patterns, which in turn may differ in complexity.

Example 6. The set 1^+0^* is a verification pattern for *some* as its permutation closure is the set of all strings that contain at least one instance of 1. The set $0^*1^+0^*$ is also a verification pattern, but it is more complex. The verification pattern 1^+0^* is SL-2 as it is generated by the positive grammar $\{\rtimes 1, 11, 10, 00, 1\ltimes, 0\ltimes\}$. But $0^*1^+0^*$ is not SL: for any choice of k, $f_k(\rtimes^{k-1}0^k\ltimes^{k-1}) \subseteq f_k(\rtimes^{k-1}0^k\ 1\ 0^k\ltimes^{k-1})$, and thus every SL-$k$ grammar that generates $0^k\ 1\ 0^k \in 0^*1^+0^*$ also generates $0^k \notin 0^*1^+0^*$. Nonetheless, the existence of an SL-2 verification pattern for *some* entails that this quantifier is SL-2 verifiable.

As we will see next, the shift from quantifier languages to verification patterns greatly alters the complexity landscape and brings the complexity of *most* in line with other MSDQs.

3.2 SL-2 verification patterns cover the typology

The class of SL-2 string languages is extremely restricted in terms of its expressivity. For example, many phenomena in phonology are strictly local, but not all of them are strictly 2-local.

Example 7. Intervocalic voicing can be construed as a phonotactic constraint against sequences where a voiceless sound occurs immediately between two vowels. This is SL-3: the set of permissible trigrams does not contain any xyz such that x and z are vowels and y is a voiceless sound. But it is not SL-2 because, say, illicit *asola* only contains bigrams that also occur in *as* or *sola*, neither one of which violates intervocalic voicing. ⌟

Quantifier	1	0
$\lvert A\rvert = 0$		
every	✓	
no		✓
always true	✓	✓

Table 2: All four SL-1 grammars over $\{1, 0\}$ and the quantifiers that they generate verification patterns for

The verification patterns for MSDQs, however, all seem to be SL-2.

Consider first the class of SL-1 string languages over $\Sigma := \{0, 1\}$. For SL-1 languages, we do not need to add edge markers to the alphabet because for $k = 1$, $\rtimes^{k-1}s\ltimes^{k-1} = \rtimes^{1-1}s\ltimes^{1-1} = \rtimes^0 s\ltimes^0 = s$ for every string s. Hence there are only four distinct SL-1 grammars over this alphabet, each one a subset of Σ. The empty grammar allows nothing at all and generates the empty language. The grammar $\{0, 1\}$ allows everything and thus generates Σ^*. Both are pathological from a linguistic perspective. The empty language is a verification pattern for the quantifier that requires $|A| = 0$ irrespective of how B is chosen, which is unlike any generalized quantifier in natural language. Similarly, Σ^* is the verification pattern of a tautological quantifier Q with $Q(A, B) = 1$ for all A and B. The two remaining grammars are $\{1\}$ and $\{0\}$, which are more interesting. The former generates all members of 1^*, and the latter generates all members of 0^*. These are the verification patterns for *every* (without existential import) and *no*, respectively (since these verification patterns are already closed under permutation, we have $V_{every} = L_{every}$ and $V_{no} = L_{no}$). The class of SL-1 string languages over $\{0, 1\}$ thus already furnishes verification patterns for *every* and *no* (see also Tab. 2), and thus *every* and *no* are both SL-1 verifiable.

The space of SL-2 grammars is significantly larger. There are $4^2 = 16$ distinct bigrams in Σ_E. Even though 7 of them can never be members of $f_2(\rtimes s\ltimes)$ for any string s (e.g. $0\rtimes$, $\ltimes 1$, and $\ltimes\ltimes$), this still leaves us with 9 useful bigrams, and hence $2^9 = 512$ distinct grammars. The total number of verification patterns is smaller because some grammars generate the same string language, for instance $\{\rtimes\ltimes\}$ and $\{\rtimes\ltimes, \rtimes 1\}$. Nonetheless the range of options is too large to discuss all of them here. Instead, I only consider grammars where strings must always start with 1 (the grammar contains $\rtimes 1$ but not $\rtimes 0$ or $\rtimes\ltimes$) and strings can end in 1 or 0 (the grammar contains both $1\ltimes$ and $0\ltimes$). This leaves us with 16 distinct grammars which differ only with respect to which of the following four bigrams they contain: 11, 10, 01, 00. Surprisingly, this is enough to generate the verification patterns for all MSDQs that aren't SL-1 verifiable, including *most*.

Table 3 lists each grammar and the quantifier that corresponds to the generated verification pattern. Out of those sixteen grammars, five generate verification patterns for unnatural quantifiers: 1), 3), 4), 5), and 11). In addition, all four of 2), 7), 8) and 14) generate the same verification pattern, which is for *every* with existential import (due to the mandatory 1 at the beginning of each string). Finally, 13) and 16) generate distinct verification patterns — $1^{+}0^{*}$ and $1\{0,1\}^{*}$, respectively — but since both impose no requirements beyond the presence of at least one 1, they are both verification patterns for *some*. The remaining verification patterns are for five distinct generalized quantifiers: *all except for at most one*, *half*, *exactly one*, *at most half*, and crucially, *at least half/most*. So even though we saw in Sec. 2.1 that L_{most} is much more complex than L_{every}, L_{no}, and L_{some}, their verification patterns are of similar complexity. The property that holds of every attested MSDQ Q is that V_Q is SL-2. In other words, every attested MSDQ is SL-2 verifiable.

3.3 most = at least half?

The reader may object that the discussion so far incorrectly conflates *most* with *at least half*. The truth-conditional definition of *most* is usually given as $|A \cap B| > |A - B|$ rather than $|A \cap B| \geq |A - B|$; or equivalently, as $|A \cap B| > \frac{1}{2}|A|$ rather than $|A \cap B| \geq \frac{1}{2}|A|$. The standard definition thus equates *most* with *more than half* rather than *at least half*. There are several responses to this challenge.

First, the verification pattern identified with *at least half*/*most* in Tab. 3 is $1^{+}(01^{+})^{*}(0)$ — the string must start with 1, may end with 1 or 0, and 1s can be followed by 1 or 0, but 0 cannot be followed by 0. If $|A|$ is odd, this pattern necessarily contains more 1s than 0. Hence the discrepancy between the verification pattern and the standard definition only arises with domains of even cardinality. But it is unclear whether the association with *at least half* rather than *more than half* is at odds with native speakers' judgments in this case. This is because native speakers generally expect *most* to indicate that $|A \cap B|$ is noticeably larger than $|A - B|$. Hence the standard definition needs to be augmented with a mechanism such as pragmatic strengthening or a theory of vagueness in order to account for the observed behavior (see Carcassi and Szymanik 2021 for a recent discussion). Once that modification is made, though, the difference between $|A \cap B| > |A - B|$ and $|A \cap B| \geq |A - B|$ becomes immaterial.

Second, it may be the case that speakers expect verification patterns to use all bigrams in the grammar. In that case, the verification pattern for *most* will always include at least one instance of 11 and thus contain more 1s than 0s. This approach will be discussed in greater detail in Sec. 4.2.

Finally, we could consider a modified verification pattern where strings can only end in 1. This, too, would ensure that there are always more 1s than 0s, and it would not change the fact that *most* has an SL-2 verification pattern. However, this undermines one advantage of SL grammars relative to finite-state automata, namely that they can easily be viewed as generators of infinite strings as long as they impose no constraints on how a string may end. As will be discussed in Sec. 4.1, this furnishes a new way to analyze statements like "most natural numbers are not a multiple of three", which are challenging for definitions based on cardinality. Requiring the verification pattern of *most* to both start and end with 1 thus addresses the minor mismatch in definitions over finite domains, but it does so at the cost of making it harder to work with infinite domains.

3.4 Fitting the typology

The shift from quantifier languages to verification patterns has revealed *most* to be no more complex than other quantifiers such as *some*. Quantifiers such as *a third of* or *an even number of*, on the other hand, are not SL-2 verifiable (and *an even number of* isn't even SL verifiable, cf. Example 6). This explains why *most* mirrors *every*, *no* and *some* in that at least some languages have morphologically simplex realizations of *most* whereas no such realizations are attested for *a third of* or *an even number of*. What distinguishes *every*, *no*, *some*, and *most* from *a third of* and *an even number of* is that the former are SL-2 verifiable.

SL-2 verifiability does not entail, though, that a quantifier can be an MSDQ. We already saw in Sec. 3.2 that the class of SL-2 verifiable quantifiers includes at least five highly unnatural quantifiers.

	Quantifier	11	10	01	00	Dead ends?	Useless bigrams?
1)	$\lvert A\rvert = \lvert A \cap B\rvert = 1$					✓	
2)	every (existential import)	✓					
3)	$1 \leq \lvert A\rvert \leq 2$ & $\lvert A \cap B\rvert = 1$		✓			✓	
4)	$\lvert A\rvert = \lvert A \cap B\rvert = 1$			✓		✓	✓
5)	$\lvert A\rvert = \lvert A \cap B\rvert = 1$				✓	✓	✓
6)	all except for at most one	✓	✓			✓	
7)	every (existential import)	✓		✓			✓
8)	every (existential import)	✓			✓		✓
9)	half $(+/-1)$		✓	✓			
10)	exactly one		✓		✓		
11)	$\lvert A\rvert = \lvert A \cap B\rvert = 1$			✓	✓	✓	✓
12)	at least half/most	✓	✓	✓			
13)	some	✓	✓		✓		
14)	every (existential import)	✓		✓	✓		✓
15)	at most half		✓	✓	✓		
16)	some	✓	✓	✓	✓		

Table 3: List of quantifiers whose verification pattern only contains strings starting with 1

This was under the additional restriction that strings must start with 1. If strings are allowed to start with 0, then many more quantifiers are SL-2 verifiable, including some that are attested but never have a morphologically simplex realization.

Example 8. The quantifier *not all* is SL-2 verifiable as its verification pattern is generated by the SL-2 grammar $\{\rtimes 0, 00, 01, 11, 0\ltimes, 1\ltimes\}$. This makes *not all* a counterpart to 14) for *every* in Tab. 3 where $\rtimes 1$ has been replaced with $\rtimes 0$. ⌟

It follows that SL-2 verifiability is a necessary property but not a sufficient one.

It is tempting, then, to look for additional restrictions that prune down the set of all SL-2 verifiable quantifiers to just those that can be realized as morphologically simplex determiners. Perhaps unsurprisingly, there are multiple options that differ slightly in what set they pick out. This is illustrated in Tab. 3 for those verification patterns that must start with 1.

If every SL-2 grammar must contain 11, this rules out all unnatural quantifiers and leaves only (several versions of) *every* and *some*, as well as *most* and *all except for at most one*.

Alternatively, one could require that only the bigrams with edge markers may be *dead ends*, i.e. bigrams that make it impossible to continue the string.

Example 9. The verification pattern for *all except for at most one* is generated by the SL-2 grammar $\{\rtimes 1, 11, 10, 1\ltimes, 0\ltimes\}$. Here 10 is a dead end. Once we encounter 10 in a string, we know that we have reached its end.

Now consider the minimally different SL-2 grammar $\{\rtimes 1, 11, 10, 00, 1\ltimes, 0\ltimes\}$, which generates a verification pattern for *some*. Here 10 is not a dead end. If one encounters 10, it is still possible for the string to continue with an arbitrary number of 0s. The only dead ends are $1\ltimes$ and $0\ltimes$. ⌟

The intuition behind the ban against dead ends is that a verification procedure should not be at risk of getting stuck before all elements have been evaluated. This requirement rules out all unnatural quantifiers and *all except for at most one*, but leaves *half* and *exactly one*. Interestingly, *half* has a simplex realization as a noun in Russian, and depending on one's semantic priors *exactly one* could be taken to be realized by the numeral *one*. One characterization of MSDQs, then, is as the class of SL-2 verifiable quantifiers whose verification patterns must start with 1 and whose SL-2 grammars must not contain dead ends (other than $1\ltimes$ and $0\ltimes$).

This is just one of many conceivable characterizations. As an illustration, Tab. 3 also indicates whether a given SL-2 grammar contains *useless* bigrams. A bigram is useless if it does not appear in any string generated by the grammar. Forbidding grammars with useless bigrams eliminates some but not all unnatural quantifiers, and it rules out several variants of *every* with existential import. This is not necessarily a good thing as it undermines some analytical options that are briefly explored in Sec. 4.5.

Yet another approach would limit the focus to just the SL-2 grammars 2) for *every*, 6) for *all except for at most one*, 12) for *most*, and 16) for *some*. These can be picked out by positing a hierarchy $11 < 10 < 01 < 00$ such that a grammar may contain a bigram y only if it also contains all x to the left of y. Monotonicity requirements of this kind seem to be common across language modules (Keenan and Comrie, 1977; Keine, 2016; Graf, 2019a, 2020; Moradi, 2020, 2021a,b). In combination with the ban against dead ends, this monotonicity requirement would only leave *every*, *no*, *some*, and *most*, but again at the cost of losing the analytical options in Sec. 4.5.

In sum, it is certainly possible to formulate additional restrictions on SL-2 verifiability to pick out specific subclasses that more closely match the attested typology. More work is needed to determine which set of restrictions is the most elegant and insightful. Even without these restrictions, though, SL-2 verifiability provides a very tight upper bound on quantifier complexity while readily accommodating a large number of natural language D-quantifiers, including all that can have morphologically simplex realizations.

4 Exploratory remarks

4.1 Claims over infinite domains

One problem of the semantic automata approach is that it represents the domain A as a string, which must be finite. Infinite domains would require the switch to ω-automata (Perrin and Pin, 2004). This issue does not arise with SL-k grammars. Since SL-k grammars determine the well-formedness of each string based on its set of k-grams, they can be easily generalized to also generate infinite strings. An infinite binary string is a mapping s from the set $\mathbb{N}$ of natural numbers into $\{\rtimes, 0, 1\}$ such that $s(n) = \rtimes$ iff $n = 0$. Infinite strings have no right edge and thus contain no right edge marker $\ltimes$, but this does not matter for the SL-2 grammars in Tab. 3 because they contain both $1\ltimes$ and $0\ltimes$ and thus put not restrictions on the end of a string.

Interestingly, this means that the meaning of *most* generalizes immediately from finite domains to infinite domains. With respect to SL-2 verifiability, *most* states that it must be possible to arrange the elements of the domain A in such a manner that 0s are never repeated. Technically this is the case for both “most natural numbers are not a multiple of three” and “most natural numbers are a multiple of three”, but the latter requires a much greater rearrangement of elements relative to the standard order of natural numbers. Under the plausible assumption that finding such a suitable rearrangement is cognitively taxing, it is not surprising that speakers are likely to consider the former statement true and the latter false.

4.2 Pragmatic strengthening

Quantifiers are subject to pragmatic strengthening. For example, *most* is usually interpreted as *most but not all*, presumably because the speaker could have said *all* instead. Pragmatic strengthening can be modeled as the requirement that the verification string must contain all the bigrams listed in the grammar (assuming the bigram is not useless and does not contain edge markers). Then a string like 111 would still be a verification pattern for *most*, but since it does not contain any instance of 10 or 01, it would also be infelicitous.[2] In terms of formal language theory, this corresponds to a step up from SL to the class of locally testable languages (McNaughton, 1974).

4.3 Modifying proportions

The proportion of 1s required by *most* can be modified by changing the locality domain. For instance, the SL-3 grammar $\{\rtimes \rtimes 1, \rtimes 11, 111, 110, 101, 011, 11\ltimes, 10\ltimes, 01\ltimes, 1 \ltimes \ltimes, 0 \ltimes \ltimes\}$ requires that the number of 1s is at least double that the number of 0s. This might be yet another instance of pragmatics going beyond the limits of SL-2 verifiability in order to strengthen the meaning of quantifiers. Perhaps the strategy could also be used to model vague quantifiers such as *many* and *few*.

4.4 Existential import

The analysis in Sec. 3 posits two different versions of *every*, one with existential import (with multiple options in Tab. 3), and one without (listed in Tab. 2). Existential import can be removed from the SL-2 grammar for a given quantifier by adding $\rtimes\ltimes$ to it. Similarly, pragmatics can add existential import by removing $\rtimes\ltimes$. The proper modeling of existential import has to be left to future work, but SL verifiability seems to be well-equipped to deal with the problem.

4.5 Typological frequency

Whereas *every* and *some* are common across languages, *no* and *most* are comparatively rare. This roughly matches the number of verification patterns we identified for each one of these quantifiers: 5

[2] This proposal requires that quantifier 6) be treated as yet another variant of *some* so that one can correctly capture the pragmatic strengthening of *some* to *some but not most/all* in cases where only one element of A is not an element of B.

for *every*, 2 for *some*, 1 for *no*, and 1 for *most*. Depending on which constraints on SL-2 grammars one adds or drops, these numbers may change significantly. Additional work is needed before a link between a quantifier's typological frequency and its number of verification patterns can be deemed plausible, but the possibility is intriguing.

4.6 Parallels to syntax

The key difference between quantifiers languages and verification patterns is that the latter express the best case complexity of a given dependency where the linear order of symbols in the string does not introduce additional complications. This is comparable to a well-known split in computational syntax that underlies *Parikh's theorem* (Parikh, 1966), the two-step approach (Morawietz, 2003; Mönnich, 2006), subregular syntax (Graf, 2022a,b), and also Minimalist syntax (Chomsky, 1995). They all observe that the complexity of syntactic dependencies is contingent on choices of linearization, recasting syntax as a system of fairly simple dependencies that interact with a complex system of linearization requirements.

Example 10. Consider the string language $(abc)^n$, which is regular. By moving all instances of c to the end of the string, we obtain the context-free language $(ab)^n c^n$ instead. If in addition we order all as before all bs, the result is the tree-adjoining language $a^n b^n c^n$. Finally, if we allow every possible permutation, then we get the MIX language, which is a 2-MCFL (Salvati, 2015). Each one of these orderings represents a marked step up in complexity. ⌟

Something similar may hold for quantifiers, with verification patterns capturing the underlying dependency imposed by quantifiers *modulo* the additional complications of actual verification in a given scenario.

4.7 The cognitive status of verification patterns

The parallel to syntax also highlights why verification patterns should not be equated with verification procedures. A verification procedure parses an input into a form that yields a verification pattern. As experimental results such as Lidz et al. (2011) and Kotek et al. (2015) arguably observe verification procedures, not verification patterns, it is not trivial to make any inferences from the former about the latter.

This again mirrors the situation in syntax: a given grammar formalism, say Minimalist grammars (Stabler, 1997, 2011a), has many different parsing algorithms ranging from CKY and Earley (Harkema, 2001) to recursive descent (Stabler, 2011b, 2013) and left-corner parsing (Stanojević and Stabler, 2018), which in turn must be combined with one of many conceivable linking theories in order to obtain predictions for human sentence processing (Kobele et al., 2013; Gerth, 2015; Graf et al., 2017; Lee, 2018; De Santo, 2020; Pasternak and Graf, 2021; Liu, 2023). Verification patterns provide a similarly rigorous approach to experimental findings. Instead of intuitive stories about the processing of quantifiers, we need I) a parsing algorithm that translates stimuli into strings matching a given verification pattern, and II) a rigorous linking theory that translates the operations of the parser into predictions about human behavior.

5 Conclusion

Among morphologically simplex quantifiers that are determiners (MSDQs), *most* is an outlier due to the complexity of its quantifier language. The picture painted by quantifier languages is misleading, though. If one does away with permutation closure and considers verification patterns instead, complexity is lowered significantly. All MSDQs have SL-2 verification patterns, and SL-2 grammars furnish several parameters that allow us to home in on just the class of typologically attested MSDQs. In addition, SL-2 patterns are extremely simple and also play a central role in phonology, morphology, and syntax, revealing quantifiers to be yet another facet of a very general piece of subregular machinery that drives language.

The approach presented in this paper is reasonably flexible and could possibly be extended to account for pragmatic strengthening, vague quantifiers and typological frequency effects, among other things. It is not limited to MSDQs, either. Future investigations of numerals, modals, and adverbial quantifiers might well confirm (or refute) the central status of SL verifiability in quantification and thus offer deep insights into how complex a meaning can packed into simplex expressions.

Acknowledgements

The work carried out for this project was supported by the National Science Foundation under Grant No. BCS-1845344. I am grateful to the three anonymous reviewers for their very detailed feedback and suggestions.

References

Phillip Burness, Kevin McMullin, and Jane Chandlee. 2021. Long-distance phonological processes as tier-based strictly local functions. *Glossa*, 6:1–37.

Fausto Carcassi and Jakub Szymanik. 2021. 'most' vs 'more than half': An alternatives explanation. In *Proceedings of the Society for Computation in Linguistics*, volume 4, pages 334–343.

Jane Chandlee. 2017. Computational locality in morphological maps. *Morphology*, 27:599–641.

Jane Chandlee. 2022. Less is more: Reexamining assumptions through the narrow focus of subregularity. *Theoretical Linguistics*, 48:205–218.

Noam Chomsky. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124.

Noam Chomsky. 1959. On certain formal properties of grammars. *Information and Control*, 2:137–167.

Noam Chomsky. 1995. *The Minimalist Program*. MIT Press, Cambridge, MA.

Noam Chomsky and Marcel-Paul Schützenberger. 1963. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, Studies in Logic and the Foundations of Mathematics, pages 118–161. North-Holland, Amsterdam.

Aniello De Santo. 2020. *Structure and Memory: A Computational Model of Storage, Gradience, and Priming*. Ph.D. thesis, Stony Brook University.

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

Sabrina Gerth. 2015. *Memory Limitations in Sentence Comprehension. A Structure-Based Complexity Metric of Processing Difficulty*. Ph.D. thesis, University of Potsdam.

Thomas Graf. 2019a. Monotonicity as an effective theory of morphosyntactic variation. *Journal of Language Modelling*, 7:3–47.

Thomas Graf. 2019b. A subregular bound on the complexity of lexical quantifiers. In *Proceedings of the 22nd Amsterdam Colloquium*, pages 455–464.

Thomas Graf. 2020. Monotonicity in syntax. In *Monotonicity in Logic and Language*, volume 12564 of *Lecture Notes in Computer Science*, pages 35–53, Berlin, Heidelberg. Springer.

Thomas Graf. 2022a. Diving deeper into subregular syntax. *Theoretical Linguistics*, 48:245–278.

Thomas Graf. 2022b. Subregular linguistics: Bridging theoretical linguistics and formal grammar. *Theoretical Linguistics*, 48:145–184.

Thomas Graf, James Monette, and Chong Zhang. 2017. Relative clauses as a benchmark for Minimalist parsing. *Journal of Language Modelling*, 5:57–106.

Martin Hackl. 2009. On the grammar and processing of proportional quantifiers: Most versus more than half. *Natural Language Semantics*, 17(1):63–98.

Kenneth Hanson. 2023a. A computational perspective on the typology of agreement. Ms., Stony Brook University.

Kenneth Hanson. 2023b. A TSL analysis of Japanese case. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2023*, pages 15–24.

Henk Harkema. 2001. *Parsing Minimalist Languages*. Ph.D. thesis, University of California.

Jeffrey Heinz. 2018. The computational nature of phonological generalizations. In Larry Hyman and Frank Plank, editors, *Phonological Typology*, Phonetics and Phonology, chapter 5, pages 126–195. Mouton De Gruyter.

Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. 2011. Tier-based strictly local constraints in phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 58–64.

Adam Jardine and Jeffrey Heinz. 2016. Learning tier-based strictly 2-local languages. *Transactions of the ACL*, 4:87–98.

Edward Keenan and Dag Westerståhl. 1996. Generalized quantifiers in linguistics and logic. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 837–894. North-Holland, Amsterdam.

Edward L. Keenan and Bernard Comrie. 1977. Noun phrase accessiblity and universal grammar. *Linguistic Inquiry*, 8:63–99.

Stefan Keine. 2016. *Probes and Their Horizons*. Ph.D. thesis, University of Massachusetts, Amherst.

Gregory M. Kobele, Sabrina Gerth, and John T. Hale. 2013. Memory resource allocation in top-down Minimalist parsing. In *Formal Grammar: 17th and 18th International Conferences, FG 2012, Opole, Poland, August 2012, Revised Selected Papers, FG 2013, Düsseldorf, Germany, August 2013*, pages 32–51, Berlin, Heidelberg. Springer.

Hadas Kotek, Yasutada Sudo, and Martin Hackl. 2015. Experimental investigations of ambiguity: The case of *most*. *Natural Language Semantics*, 23:119–156.

Dakotah Lambert and James Rogers. 2020. Tier-based strictly local stringsets: Perspectives from model and automata theory. In *Proceedings of the Society for Computation and Linguistics*, volume 3, pages 330–337.

So Young Lee. 2018. A minimalist parsing account of attachment ambiguity in English and Korean. *Journal of Cognitive Science*, 19:291–329.

Jeff Lidz, Paul Pietroski, Tim Hunter, and Justin Halberda. 2011. Interface transparency and psychosemantics of *most*. *Natural Language Semantics*, 19:227–256.

Lei Liu. 2023. Processing advantages of end-weight. *Proceedings of the Society for Computation in Linguistics*, 6:250–258.

Connor Mayer. 2021. Capturing gradience in long-distance phonology using probabilistic tier-based strictly local grammars. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2021*, pages 39–50.

Kevin McMullin. 2016. *Tier-Based Locality in Long-Distance Phonotactics: Learnability and Typology*. Ph.D. thesis, University of British Columbia.

Kevin McMullin and Gunnar Ólafur Hansson. 2015. Long-distance phonotactics as tier-based strictly 2-local languages. In *Proceedings of AMP 2014*.

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

Uwe Mönnich. 2006. Grammar morphisms. Ms. University of Tübingen.

Sedigheh Moradi. 2020. Morphosyntactic patterns follow monotonic mappings. In *Monotonicity in Logic and Language*, pages 147–165, Berlin, Heidelberg. Springer Berlin Heidelberg.

Sedigheh Moradi. 2021a. A formal restriction on gender resolution. In *All Things Morphology: Its Independence and Its Interfaces*, pages 41–54. John Benjamins, Amsterdam.

Sedigheh Moradi. 2021b. *Monotonicity in Morphosyntax*. Ph.D. thesis, Stony Brook University.

Frank Morawietz. 2003. *Two-Step Approaches to Natural Language Formalisms*. Walter de Gruyter, Berlin.

Rohit Parikh. 1966. On context-free languages. *Journal of the Association for Computing Machinery*, 13:570–581.

Robert Pasternak and Thomas Graf. 2021. Cyclic scope and processing difficulty in a Minimalist parser. *Glossa*, 6:1–34.

Dominique Perrin and Jean-Éric Pin. 2004. *Infinite Words. Automata, Semigroups, Logic and Games*. Elsevier, Amsterdam.

Stanley Peters and Dag Westerståhl. 2006. *Quantifiers in Language and Logic*. Oxford University Press.

Sylvain Salvati. 2015. MIX is a 2-MCFL and the word problem in $\mathbb{Z}^2$ is captured by the IO and the OI hierarchies. *Journal of Computer and System Sciences*, 81:1252–1277.

Edward P. Stabler. 1997. Derivational Minimalism. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics*, volume 1328 of *Lecture Notes in Computer Science*, pages 68–95. Springer, Berlin.

Edward P. Stabler. 2011a. Computational perspectives on Minimalism. In Cedric Boeckx, editor, *Oxford Handbook of Linguistic Minimalism*, pages 617–643. Oxford University Press, Oxford.

Edward P. Stabler. 2011b. Top-down recognizers for MCFGs and MGs. In *Proceedings of the 2nd Workshop on Cognitive Modeling and Computational Linguistics*, pages 39–48.

Edward P. Stabler. 2013. Two models of minimalist, incremental syntactic analysis. *Topics in Cognitive Science*, 5:611–633.

Miloš Stanojević and Edward Stabler. 2018. A sound and complete left-corner parser for Minimalist grammars. In *Proceedings of the 8th Workshop on Cognitive Aspects of Computational Language Learning and Processing*, pages 65–74.

Shane Steinert-Threlkeld and Thomas F. Icard. 2013. Iterating semantic automata. *Linguistics and Philosophy*, 36(2):151–173.

Johan van Benthem. 1986. Semantic automata. In *Essays in Logical Semantics*, pages 151–176. Springer, Dordrecht.