A Subregular Bound on the Complexity of Lexical Quantifiers

Thomas Graf

Stony Brook University, Stony Brook, NY, USA
mail@thomasgraf.net

Abstract

Semantic automata were developed to compare the complexity of generalized quantifiers in terms of the string languages that describe their truth conditions. An important point that has gone unnoticed so far is that these string languages are remarkably simple for most quantifiers, in particular those that can be realized by a single lexical item. Whereas complex quantifiers such as *an even number of* correspond to specific regular languages, the lexical quantifiers *every*, *no*, *some* as well as numerals do not reach this level of complexity. Instead, they all stay close to the bottom of the so-called subregular hierarchy. What more, the class of tier-based strictly local languages provides a remarkably tight characterization of the class of lexical quantifiers. A significant number of recent publications have also argued for the central role of tier-based strict locality in phonology, morphology and syntax. This suggests that subregularity in general and tier-based strict locality in particular may be a unifying property of natural language across all its submodules.

1 Introduction

Generalized quantifiers (GQs) have been a fruitful topic for mathematical investigation since Barwise and Cooper (1981) and Keenan and Faltz (1985). Among the many techniques to study GQs (see Peters and Westerståhl, 2006), semantic automata (van Benthem, 1986; Clark, 2001; Steinert-Threlkeld and Icard, 2013) are noteworthy because their grounding in formal language theory provides a bridge to phonology and syntax, which have also been studied from this perspective. However, computational phonology and syntax have recently undergone a shift towards particularly weak classes of formal languages that belong to the *subregular hierarchy* (see Graf 2018, Heinz 2018, and references therein). In this paper, I argue that the subregular hierarchy also provides a fruitful perspective on GQs.

The semantic automata approach characterizes GQs via string languages that encode their truth conditions. It is known that many GQs have string languages that belong to the class of regular languages. This result can be strengthened significantly, though, as many of the most common quantifiers belong to the subregular class of *tier-based strictly local* string languages (TSL; Heinz et al., 2011), which plays a central role in phonology and syntax. This includes *every*, *some*, *no*, numerals, *not all*, and *all but three*, but not *most* or *half*. In addition, TSL provides a new perspective on a long-standing puzzle in morphosemantics: Across all languages, only a small number of GQs are ever realized as a single lexical item. These *lexical quantifiers* include *every*, *some*, *no*, and numerals, to the exclusion of *not all*, *all but three*, or *an even number of*. Why only some GQs can be lexicalized in this manner has remained mysterious. TSL establishes a marked difference between these two groups, a difference that is connected to the semantically fertile notion of monotonicity. Lowering the bound of GQs to TSL thus has two major payoffs. It establishes a computational parallel between semantics on the one hand and syntax and phonology on the other, and it explains why only some GQs can be realized as a single lexical item.

The paper is organizes as follows. I start with a brief introduction to generalized quantifiers and how they can be represented as string languages (§2). This section can be skipped by readers who are already familiar with semantic automata (but they might still want to take a look at the table in (1)). After that, I discuss TSL and show that this class subsumes many simple GQs while excluding more complicated ones like *an even number of* (§3). I then use montonicity to define an even more restricted subclass of TSL, and I show that this subclass provides a tight approximation of the class of GQs that can be spelled-out as a single lexical item.

2 Generalized Quantifiers as String Languages

A GQ is a function that maps one or more relations to truth values. In linguistics, the focus of research has been mostly on quantifiers of type $\langle 1\rangle$ and $\langle 1,1\rangle$. A type $\langle 1\rangle$ quantifier maps a unary relation, i.e. a set, to a truth value. Examples are *everybody* or *nobody*, but adverbs such as *always* may also be viewed as type $\langle 1\rangle$ quantifiers. Type $\langle 1,1\rangle$ quantifiers, on the other hand, take two sets as input and return a truth value. The prototypical quantifiers of English belong to this category: *every*, *some*, and *no*. But among its members are also *most*, *an even number of*, and numerals like *five*, *at most five*, and *at least five*. In the sentence *at most five dragons are chainsmokers*, the $\langle 1,1\rangle$ quantifier *at most five* takes the set of dragons and the set of chainsmokers as its first and second argument, respectively, and returns true iff the intersection of the two sets contains at most five elements. Due to the prominence of $\langle 1,1\rangle$ quantifiers across natural languages, they are the central object of study in GQ theory (another reason is that type $\langle 1\rangle$ quantifiers may be regarded as a special case of type $\langle 1,1\rangle$ quantifiers where the first argument is fixed). This paper will also limit itself to $\langle 1,1\rangle$ quantifiers, focusing exclusively on those that are determiners.

Given a type $\langle 1,1\rangle$ quantifier Q and sets A and B, $Q(A,B)$ will be either true ($=1$) or false ($=0$), depending on the definition of Q. For example, $\mathit{every}(A,B)$ is true iff $A\subseteq B$. So *every cat is a mammal* is true because the set of cats is a subset of the set of mammals, wherefore $\mathit{every}(\mathit{cat},\mathit{mammal})$ holds. On the other hand, *no cat is a mammal* is false because $\mathit{no}(A,B)$ holds iff $A\cap B=\emptyset$. The truth value of a $\langle 1,1\rangle$ quantifier Q is thus contingent on the set relations that hold between A and B. For natural language quantifiers, additional properties hold that ensure that the meaning of a quantifier can be represented solely in terms of which members of A are members of B. This is the key to measuring GQ complexity via string languages. The basic idea is to convert the relation between A and B into a string of 0s and 1s so that Q can be viewed as a formal language of such strings.

Since the idea of representing a GQ as a string language is fairly abstract, a certain amount of rigor is required to avoid misinterpretation. Given some arbitrary enumeration e of all the elements of A, f_B^A is a total function that converts e into a string over the alphabet $\{0,1\}$. Every element of A that also belongs to B is replaced by 1, all others by 0. So with $A:=\{a,b,c\}$ and $B:=\{a,d\}$, the enumeration $e:=\mathit{bac}$ of A would be rewritten as 010 by f_B^A. For a more concrete example, consider f_B^A with B the set of all males and A the set of all US presidents up to and including 2019. This function produces a string of length 45 where every symbol is 1 because every member of the set of US presidents is also in the set of male individuals. If A is the set of all humans, on the other hand, f produces a string with approximately 7 billion symbols, more than half of which are 0 because only 49% of the current world population are male. The order of 0s and 1s in this string depends on how one enumerates A, which is ultimately immaterial as any enumeration is a valid choice.

For any two (countably infinite) sets A and B, then, f_B^A encodes the relation between the

two as a (usually non-unique and possibly infinite) string over 0 and 1. I will call this a *binary string over A and B*. A GQ Q, in turn, is equivalent to a language $L(Q)$ of binary strings. Throughout this paper, I refer to such sets of binary strings as *quantifier languages*.[1]

Definition 1 (Binary Strings)**.** *Let A and B be two (countably infinite) sets. The total function f_B^A maps each enumeration e of A to a (possibly infinite) string of 0s and 1s:*

$$f_B^A(e) := \begin{cases} f_B^A(a) \cdot f_B^A(e') & \text{if } e = a \cdot e', \text{ and } a \in A, \text{ and } e' \text{ is not the empty string } \varepsilon \\ 1 & \text{if } e \in A \cap B \\ 0 & \text{otherwise} \end{cases}$$

Here $\cdot$ denotes string concatenation. We call s a binary string of A under B *iff there is some enumeration e of A such that $s = f_B^A(e)$.*

Definition 2 (Quantifier Language)**.** *Let Q be a type $\langle 1,1 \rangle$ quantifier. Then its* quantifier language $L(\mathrm{Q})$ *is the unique set such that for all sets A and B and (possibly infinite) binary string s of A under B, it holds that $s \in L(\mathrm{Q})$ iff $Q(A,B)$ is true.*

Consider once more the case of *every*, and recall that *every*(A,B) is true iff $A \subseteq B$. If $A \subseteq B$, then every binary string of A under B contains only 1s, and never any 0s. On the other hand, $A \subsetneq B$ implies that there is at least some $a \in A$ that is not a member of B, and this a will be rewritten as 0 by f_B^A. This shows that the binary strings that encode a subset relation between A and B are exactly those that contain only 1s. The quantifier language $L(\mathit{every})$ thus consists of all strings, and only those, that do not contain any 0s, and *every*(A,B) is true iff f_B^A produces a binary string of this form. The same line of reasoning shows that $L(\mathit{no})$ consists of all and only those strings that contain only 0, while $L(\mathit{some})$ consists of exactly those strings with at least one 1.

The quantifier languages of *every*, *no*, *some*, *not all*, and numerals can be succinctly described in terms of their numerosity requirements on 1 and 0. The left table in (1) shows how this is done, with $|s|_0$ and $|s|_1$ denoting the number of 0s and 1s in a binary string s, respectively. The right table shows that other GQs have string languages for which the conditions are trickier to state (keep in mind that proportional GQs are only well-defined if A is finite).

(1)

Quantifier	Language
every	$\lvert s\rvert_0 = 0$
no	$\lvert s\rvert_1 = 0$
some	$\lvert s\rvert_1 \geq 1$
not all	$\lvert s\rvert_0 \geq 1$
at least n	$\lvert s\rvert_1 \geq n$
at most n	$\lvert s\rvert_1 \leq n$
exactly n	$\lvert s\rvert_1 = n$

Quantifier	Language
between m and n	$m \leq \lvert s\rvert_1 \leq n$
an even number	$\lvert s\rvert_1 \mod 2 = 0$
half	$\lvert s\rvert_1 = \lvert s\rvert_0$
most	$\lvert s\rvert_1 > \lvert s\rvert_0$
at least one third	$3\lvert s\rvert_1 \geq \lvert s\rvert_0 + \lvert s\rvert_1$

The fact that some GQs have simpler descriptions of their string languages than others already suggests that certain complexity differences exist between them. Formal language theory provides the tools to state this explicitly in computational terms.

[1] Semantic automata theory usually assumes that all binary strings are finite because automata only generate finite strings. But this is not a necessity. The theory of ω-automata (Perrin and Pin, 2004), for instance, allows for languages with infinite strings. More importantly, the grammar-based specification adopted in this paper is completely agnostic about whether strings are finite or infinite, as long as they are countably infinite.

3 Subregular Quantifier Languages: TSL

We just saw how the semantic automata approach recasts GQs as sets of binary strings. This has made it possible to establish important complexity differences between quantifiers. But as we will see next, the semantic automata approach has largely relied on very coarse classes that can hide important complexity differences (§3.1). The subregular notion of tier-based strict locality refines the existing picture in an insightful manner. I first describe the class in intuitive terms (§3.2) and show that it covers a wide range of simple GQs (§3.3) while excluding more complex ones (§3.4). The formal definition is provided in §3.5.

3.1 From Regular to Subregular

Building directly on the Chomsky hierarchy (Chomsky, 1956), work on semantic automata has largely been concerned with the question which quantifier languages belong to the weakest level of that hierarchy, the *regular string languages*. It turns out that almost all natural language quantifiers are regular, including: *every*, *no*, *some*, *not all*, *at least 5*, *between 5 and 17*, and *an even number*. But proportional quantifiers like *most*, *half*, or *at least one third* are not regular — they have *context-free* quantifier languages.

While this is an important finding, it does not fully line up with the intuitive split between simple and complex quantifier languages in (1). In particular, the bifurcation into regular and context-free GQs suggests that *an even number* is just as simple as *some* because both are regular, which seems counter-intuitive.[2] It also misses an important typological fact about GQs: across languages, *every*, *no*, *some*, and numerals are the only GQs with regular quantifier languages that can be spelled out as a single lexical item, whereas *not all* and *an even number* always require multi-word expressions (Paperno, 2011). This is unexpected if all these quantifiers are regular and hence equally complex. The Chomsky hierarchy therefore provides an incomplete picture at best.

However, it has long been known that the class of regular languages can be decomposed into an elaborate hierarchy of subclasses (Schützenberger, 1965; McNaughton and Papert, 1971; Pin, 1997). This *subregular hierarchy* has recently found numerous applications in phonology, morphology, and syntax (Graf, 2018; Heinz, 2018). The class of *tier-based strictly local* languages (TSL; Heinz et al., 2011) has turned out to be particularly useful in all three domains. Perhaps it isn't too surprising, then, that TSL can also address the issues raised above.

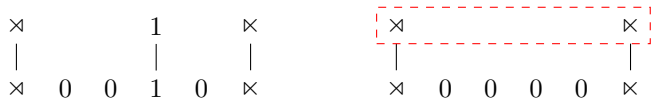
3.2 Tier-Based Strictly Local String Languages

TSL is an intuitively appealing class that is directly inspired by the notion of autosegmental tiers in phonology (Goldsmith, 1976). First one identifies a set of symbols that serves as the tier alphabet T. Given a string s, every symbol in s that belongs to T projects onto a dedicated tier. This tier is then checked for well-formedness against a finite list S of forbidden substrings (or *n-grams*). A string is well-formed iff its tier contains no forbidden substrings.

The formal definition of TSL is relegated to §3.5 at the end of this section. Instead, let us work through a concrete example by showing that $L(some)$ is TSL. Remember that $L(some)$ consists of all binary strings that contain at least one 1. Hence we have $0010 \in L(some)$ but $0000 \notin L(some)$. We can capture this with a TSL grammar that I) projects a tier containing all 1s but no 0s, and II) requires this "1-tier" to be non-empty. This is illustrated below, with

[2] It is common practice to further distinguish between simple and complex quantifier languages based on the size of the semantic automata that generate these languages. But this makes even more counter-intuitive predictions because the automaton for *an even number* is smaller than the automaton for *at least three*.

⋊ and ⋉ as dedicated symbols for the left and right string/tier edge, respectively. Using these symbols, we can rule out empty tiers by banning the substring ⋊⋉.



The left string is well-formed because ⋊⋉ is not a substring of the tier, thanks to the presence of the symbol 1. The right string, on the other hand, is illicit due to its tier lacking such a symbol and thus containing the forbidden bigram ⋊⋉.

The TSL grammar for *some* can be extended to any numeral. For instance, replacing the bigram ⋊⋉ by the 4-gram 1111 will block all binary strings with four or more 1s, as illustrated below. In other words, this new grammar defines the quantifier language for *at most 3*.

For *at least 3*, one has to rule out tiers with less than three 1s via the following list of forbidden n-grams: ⋊⋉, ⋊1⋉, ⋊11⋉. Any tier that contains at least one of those three substrings is ill-formed, and so is the binary string that the tier was projected from.

Adding the 4-gram 1111 to the list above yields a TSL grammar that generates L(*exactly 3*). If instead of 1111 we had added the 5-gram 11111, the resulting language would have been L(*between 3 and 4*).

We see, then, that TSL is powerful enough to handle *some* as well as numerals and related quantifiers.

3.3 Varying the Tier Alphabet

All the quantifiers in the previous section are modeled by TSL grammars with tier alphabet {1}. But {0}, {0, 1}, and the empty set are also valid choices for the tier alphabet. By altering the tier alphabet, even more GQs can be described with TSL.

Changing the tier alphabet to {0} allows for the expression of some other generalized quantifiers.

(2)

Forbidden Substrings	**Quantifier**
⋊⋉	not all
⋊⋉, 00	all but one
⋊⋉, ⋊0 ⋉, 000	all but two
⋮	

TSL grammars with an empty tier alphabet, on the other hand, are useless. An empty tier alphabet will produce an empty tier for every string, which means that the grammar cannot make any distinctions between strings.

This leaves us with a tier alphabet of $\{0, 1\}$. In this case, the tier is always identical to the binary string it is projected from. While this may seem useless, too, it is actually necessary to capture $L(\textit{every})$ and $L(\textit{no})$ with TSL — every symbol projects, and the tier must not contain the substring 0 for $L(\textit{every})$ or the substring 1 for $L(\textit{no})$. Projecting only 0 or only 1 would not do in this case as every symbol in the string has to be inspected by the grammar. In sum, many GQs have TSL quantifier languages, although the tier alphabet may differ between them: *every*, *some*, *no*, *not all*, *all but 3*, and numerals.

3.4 Examples of Quantifiers that are not TSL

There are also many generalized quantifiers that are not TSL. Since every TSL language is also a regular language, proportional quantifiers are not TSL by virtue of not being regular. Moreover, some quantifiers are regular but not TSL. The most prominent example is *an even number*, for reasons that are easy to verify.

Every proposed TSL grammar for $L(\textit{an even number})$ must have $\{1\}$ as its tier alphabet. Not projecting 1 makes it impossible to ensure that the binary string contains an even number of 1s, and projecting 0 only adds irrelevant material to the tier. Assume, then, that only 1 is projected. In that case, well-formed binary strings will have tiers of even length, whereas ill-formed binary strings have tiers of odd length. But a finite list of forbidden substrings cannot distinguish 1-tiers of even length from 1-tiers of odd length. The reasoning is as follows: In order to rule out a 1-tier of odd length, at least one of its substrings must be illicit. Since the list of forbidden substrings is finite, there is a longest forbidden substring s with k 1s in a row. Now take some 1-tier of even length $l > k + 2$ (we add 2 to account for $\rtimes$ and $\ltimes$). This tier should be well-formed, but as it contains s as a substring, it is mistakenly ruled out by the TSL grammar. Hence a TSL grammar for $L(\textit{an even number})$ either overgenerates by permitting some odd strings, or it forbids some even strings and thus undergenerates.

That TSL grammars cannot handle $L(\textit{an even number})$ is a welcome result. It shows that the step down from regular to TSL paints a more detailed picture of GQ complexity.

3.5 Formal Summary

With the intuition behind TSL firmly established, we can finally turn to the formal definition.

Definition 3 (Tier-Based Strictly Local)**.** *A* tier-based strictly local *grammar G over alphabet Σ is a pair $\langle S, T\rangle$, where $T \subseteq \Sigma$ is a tier alphabet and S is a finite set of (finite) strings over T. We say that G is* tier-based strictly n-local *(TSL-n) iff the length of each string in S is at most n.*

Given a string s over Σ, its T-tier is $\rtimes T(s) \ltimes$, where $\rtimes, \ltimes \notin \Sigma$ are distinguished and $T(s)$ is given by:

$$T(s) := \begin{cases} T(u) \cdot T(s') & \text{if } u \in \Sigma \text{ and } s = u \cdot s' \\ s & \text{if } s \in T \\ \varepsilon & \text{otherwise} \end{cases}$$

Here ε denotes the empty string — for all strings u over Σ, $u \cdot \varepsilon = \varepsilon \cdot u = u$. The T-tier is well-formed iff none of its substrings are members of S. The language $L(G)$ of strings generated by G contains all strings whose T-tier is well-formed, and only those. A string set is tier-based strictly local iff there is a tier-based strictly local grammar that generates it.

Quantifier	String Condition	Complexity	Tier	Forbidden
every	$\|s\|_0 = 0$	TSL	$\{0,1\}$	$\{0\}$
no	$\|s\|_1 = 0$	TSL	$\{0,1\}$	$\{1\}$
some	$\|s\|_1 \geq 1$	TSL	$\{1\}$	$\{\rtimes\ltimes\}$
not all	$\|s\|_0 \geq 1$	TSL	$\{0\}$	$\{\rtimes\ltimes\}$
(at least) n	$\|s\|_1 \geq n$	TSL	$\{1\}$	$\{\rtimes 1^k \ltimes\}_{k<n}$
(at most) n	$\|s\|_1 \leq n$	TSL	$\{1\}$	$\{1^{n+1}\}$
(exactly) n	$\|s\|_1 = n$	TSL	$\{1\}$	$\{1^{n+1}, \rtimes 1^k \ltimes\}_{k<n}$
between m and n	$m \leq \|s\|_1 \leq n$	TSL	$\{1\}$	$\{1^{n+1}, \rtimes 1^k \ltimes\}_{k<m}$
all but n	$\|s\|_0 = n$	TSL	$\{0\}$	$\{0^{n+1}, \rtimes 0^k \ltimes\}_{k<n}$
an even number	$\|s\|_1 = 2n, n \geq 0$	regular		
half	$\|s\|_1 = \|s\|_0$	context-free		
most	$\|s\|_1 > \|s\|_0$	context-free		
at least one third	$3\|s\|_1 \geq \|s\|_0 + \|s\|_1$	context-free		

Table 1: String complexity for some generalized quantifiers in English

We now have a more fine-grained view of quantifier complexity, which is summarized in Tab. 2. The table makes it very clear that TSL encompasses most of the quantifiers that linguists are inclined to consider natural. TSL gives us the desired complexity difference between those quantifiers and *an even number*. However, TSL itself still does not explain the typological fact that among all GQ with regular quantifier languages, only some can ever be spelled out as a single lexical item. In the next section, I show that TSL does in fact provide an answer to this puzzle once we look more carefully at the tier projection mechanism.

4 Lexical Quantifiers are Monotonic TSL

Our discussion so far has yielded a more refined classification of type $\langle 1,1 \rangle$ quantifiers according to their automata-theoretic complexity. While this is a novel result, it is not particularly useful in and of itself. That so many GQs are TSL is intriguing considering that this class is also commonly encountered in phonology, morphology, and syntax. But that still leaves us without any concrete empirical ramifications. In particular, there still seems to be no complexity difference between lexical quantifiers like *every* and multi-word quantifiers like *not all*. Why, then, is it that there is no known language that realizes *not all* as a single lexical item? The answer may lie in a property that is all too familiar to semanticists: monotonicity.

4.1 Monotonicity of Tier Alphabets

Consider once more the TSL quantifier languages in Tab. 1, and pay close attention to their tier alphabet. All lexical quantifiers use either $\{1\}$ or $\{0,1\}$, but never $\{0\}$. This sets them apart from quantifiers like *not all* and *all but 3*, which only project 0. The difference between the tier alphabets $\{1\}$ and $\{0,1\}$ on the one hand and $\{0\}$ on the other is that the former's characteristic functions are *monotonically increasing*.

Definition 4. *Let A and B be two sets that are partially ordered by $\leq_A$ and $\leq_B$, respectively. Then a function f from A to B is* monotonically increasing *iff $x \leq_A y$ implies $f(x) \leq_B f(y)$ for all $x, y \in A$.*

Each tier alphabet T can be regarded as a characteristic function from the string alphabet Σ into the Boolean algebra **2** of truth values, with $0 < 1$. But in the case of quantifier languages, the only alphabet symbols are 0 and 1, which are truth values. Consequently, the characteristic function of T is a mapping from **2** to **2**. The tier alphabets $\{1\}$ and $\{0, 1\}$ both correspond to monotonically increasing functions. With $\{1\}$, $0 \leq 1$ and $f(0) = 0 \leq 1 = f(1)$. With $\{0, 1\}$, $f(0) = 1 \leq 1 = f(1)$. But with $\{0\}$, we have $0 \leq 1$ yet $f(0) = 1 \not\leq 0 = f(1)$. Once we distinguish between arbitrary TSL languages and those with a monotonically increasing quantifier language, we get a much tighter approximation of the class of GQs that can be expressed by a single lexical item.

Definition 5. *A quantifier language L is* monotonic TSL *iff there is TSL grammar $G := \langle S, T \rangle$ such that $L = L(G)$ and the characteristic function of T is a monotonically increasing function from* **2** *to* **2**.

monotonic TSL	<	TSL	<	regular	<	context-free
every		*not all*		*an even number*		*most*
no		*all but n*				*half*
some						*at least one third*
(at most) n						
(at least) n						
(exactly) n						
between m and n						

Table 2: Complexity of natural language quantifier languages

Proposition 1. *If a GQ has a regular quantifier language, then this quantifier can be spelled out as a single lexical item only if its quantifier language is monotonic TSL.*

The restriction to monotonic TSL is not only natural from the perspective of semantics, where monotonicity is known to play a significant role, it is also a formal counterpart to the intuition that set membership (encoded by 1) is a simpler concept than non-membership (encoded by 0).

4.2 Open Ends

Proposition 1 is a new semantic universal on the phonetic exponents of GQs, and it marks a significant step towards a deeper understanding of the morphosemantics of quantifiers. That said, there are several loose ends.

First of all, the proposition only identifies a necessary condition, but not a sufficient one. There are GQs with monotonic TSL quantifier languages that nonetheless are never realized as a single lexical item. Table 2 already provides *between m and n* as one example of this. But there are also much more abstract quantifiers in this class. For instance, one can generalize *at least n* to a grammar with tier alphabet $\{0, 1\}$ that requires every string to be at least n symbols long. This corresponds to a quantifier Q such that $Q(A, B)$ is true iff A has at least n members. Monotonic TSL is still too large a class, and additional restrictions will be needed to identify the subclass of monotonic TSL quantifier languages that only encode quantifiers that are I) possible natural language quantifiers, and II) can be expressed by a single lexical item.

In addition, the restriction of Prop. 1 to GQs with regular string languages only serves the purpose of excluding *most* and *half*. That these two quantifiers can be realized as a single lexical

item is even more puzzling now that our subregular analysis has widened the complexity gap between them and the other GQs. There are multiple answers to this that need to be explored in detail. One solution is to follow Hackl (2009) in assuming that *most* (and possibly *half*) is actually not a single lexical item. But this would require a very specific definition of what it means to be a single lexical item, which poses the risk of circular reasoning. Alternatively, one could reassess the complexity of *most* and *half*: these two quantifiers are context-free only if one is not allowed to impose any order on the binary strings.

If quantifier languages need not be closed under permutation, then *most* and *half* are both monotonic TSL. Suppose that we have a monotonic TSL grammar with tier alphabet $\{0, 1\}$ and the following constraints on the tier: I) do not start with 0, II) do not end with 0, III) do not contain the substring 00. This only permits binary strings where every 0 is preceded by at least one 1 and followed by at least one 1. In other words, it holds for each one of these strings that more than half of its symbols are 1s. This is exactly the definition of *most*. One could speculate that this reliance on string orders might be a formal analog to match-up heuristics that seem to be used in verification tasks involving *most*. It remains to be seen whether this view of *most* is viable.[3]

5 Conclusion

I have argued that the semantic automata approach stands to gain a lot from incorporating recent work on subregular complexity. Not only does it allow for a more fine-grained classification of quantifiers, it also establishes computational parallels between phonology, morphology, syntax, and semantics, and it brings us closer to an explanation why some quantifiers can be realized as a single lexical item while others are always multi-word constructions.

I hope that this paper will serve as the starting point for a larger enterprise of subregular semantics. The obvious next step would be adverbial quantifiers, but any domain is viable as long as it can be modeled in terms of string languages (cf. the string-based view of tense in Fernando 2015).

Acknowledgments

The work reported in this paper was supported by the National Science Foundation under Grant No. BCS-1845344. I would like to thank the anonymous reviewers as well as the audiences at Stony Brook's Mathematical Linguistics Reading Group and Rutger's Linguistics colloquium. They all provided valuable feedback that led to significant improvements to this paper.

References

Jon Barwise and Robin Cooper. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4:159–219, 1981.

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

[3]No comparable strategy is available for *an even number*, so this quantifier would still not be monotonic TSL. However, *not all* and *all but n* can be made monotonically TSL by requiring all 0s to occur before all 1s.

Robin Clark. Generalized quantifers and semantic automata: A tutorial. Ms., University of Pennsylvania, 2001.

Tim Fernando. The semantics of tense and aspect: a finite-state perspective. In Shalom Lappin and Chris Fox, editors, *The Handbook of Contemporary Semantic Theory, Second edition*, pages 203–236. Wiley, 2015.

John Goldsmith. *Autosegmental Phonology*. PhD thesis, MIT, 1976.

Thomas Graf. Why movement comes for free once you have adjunction. In Daniel Edmiston, Marina Ermolaeva, Emre Hakgüder, Jackie Lai, Kathryn Montemurro, Brandon Rhodes, Amara Sankhagowit, and Miachel Tabatowski, editors, *Proceedings of CLS 53*, pages 117–136, 2018.

Martin Hackl. On the grammar and processing of proportional quantifiers: Most versus more than half. *Natural Language Semantics*, 17(1):63–98, 2009. doi: 10.1007/s11050-008-9039-x.

Jeffrey Heinz. 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, 2018.

Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. Tier-based strictly local constraints in phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 58–64, 2011. URL http://www.aclweb.org/anthology/P11-2011.

Edward L. Keenan and Leonard M. Faltz. *Boolean Semantics for Natural Language*. Reidel, Dordrecht, 1985.

Robert McNaughton and Seymour Papert. *Counter-Free Automata*. MIT Press, Cambridge, MA, 1971.

Denis Paperno. Learnable classes of natural language quantifiers: Two perspectives. Ms., UCLA, 2011. URL http://paperno.bol.ucla.edu/q_learning.pdf.

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

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

Jean-Eric Pin. Syntactic semigroups. In *Handbook of Language Theory*, pages 679–764. Springer, Berlin, 1997.

M.P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

Shane Steinert-Threlkeld and Thomas F. Icard. Iterating semantic automata. *Linguistics and Philosophy*, 36(2):151–173, 2013. ISSN 1573-0549. doi: 10.1007/s10988-013-9132-6.

Johan van Benthem. Semantic automata. In *Essays in Logical Semantics*, pages 151–176. Springer, Dordrecht, 1986. ISBN 978-94-009-4540-1. doi: 10.1007/978-94-009-4540-1_8.