# The Subregular Complexity of Monomorphemic Quantifiers

Thomas Graf
*Stony Brook University*

**Abstract** Semantic automata were developed to compare the complexity of generalized quantifiers based on the complexity of the string languages that describe their truth conditions. An important point that has gone unnoticed so far is that the generated string languages are remarkably simple for monomorphemic quantifiers. Whereas complex quantifiers such as *an even number of* correspond to specific regular languages, monomorphemic *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 monomorphemic 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.

**Keywords:** computational linguistics; generalized quantifiers; semantic automata; subregular languages; typology

## 1 Introduction

Generalized quantifiers have been a fruitful topic for mathematical investigation since Barwise & Cooper (1981) and Keenan & Faltz (1985). Among the many techniques to study generalized quantifiers (see Peters & Westerståhl 2006), semantic automata (van Benthem 1986, Clark 2001, Steinert-Threlkeld & Icard 2013) are noteworthy because they do not draw from mathematical logic but rather from formal language theory. Even though there are some well-known connections between logics and formal languages — e.g. the equivalence of regular languages and monadic second-order logic (Büchi 1960) — formal language theory does recognize complexity differences for which no logical correspondent is known at the moment. Therefore the automata-theoretic approach to generalized quantifiers, albeit less common, adds important facets to the dominant perspective rooted in logic and set-theory.

At the same time, the semantic automata approach has not made full use of the richness of formal language theory. The classification schema it uses to measure the complexity of generalized quantifiers is built directly on the well-known Chomsky hierarchy (Chomsky 1956):

$$\text{regular} \subsetneq \text{context-free} \subsetneq \text{context-sensitive} \subsetneq \text{recursively enumerable}$$

While this hierarchy still forms part of the core of formal language theory, the last 50 years of research have resulted in numerous refinements. Arguably the best-known work along these lines is the search for a class between CFL and CSL that is suitable for natural language syntax (see Huybregts 1984, Joshi 1985, Shieber 1985, Culy 1985, Radzinski 1991, Seki et al. 1991, Michaelis & Kracht 1997, Kobele 2006, and references therein). But a very different enterprise has proven just as fruitful for linguistics: the articulation of a *subregular hierarchy* (Schützenberger 1965, McNaughton & Papert 1971, McNaugthon 1974, Simon 1975, Pin 1997, Ruiz et al. 1998). Rather than looking for more powerful classes beyond the realm of context-freeness, this line of work extends the Chomsky hierarchy downwards to include weaker classes. The monolithic class of regular languages is thus refined into an articulate and very fine-grained hierarchy of subclasses.

Against the larger backdrop of the Chomsky hierarchy, the classes in the subregular hierarchy are extremely weak. Yet it is precisely this lack of expressivity that has attracted the interest of computational linguists. In the last ten years, there has been a torrent of papers that argue that phonology is subregular (see Heinz 2015, Chandlee & Heinz 2016 and references therein). Subregularity has also been found to play a central role in morphology (Aksënova et al. 2016, Chandlee 2016) and syntax (Graf & Heinz 2015). One might suspect, then, that subregularity can be detected in semantics, too, and this is exactly what I argue for in this paper.

If one follows the automata-theoretic approach in characterizing generalized quantifiers via string languages that encode their truth conditions, it turns out that almost all monomorphemic quantifiers are subregular. More precisely, these quantifiers belong to the class of *tier-based strictly local* string languages (TSL; Heinz et al. 2011). TSL is a slightly enhanced variant of the $n$-gram models that are ubiquitous in computational linguistics. In TSL, an $n$-gram model is combined with a function that masks out specific symbols in a string so that non-adjacent segments can be regarded as adjacent. The only typologically attested monomorphemic quantifiers outside TSL are the proportionality quantifiers *most* and *half*. However, *most* has been argued not to be monomorphemic (Hackl 2009), and it is unclear whether *half* should be considered monomorphemic since its proportionality reading is limited to fixed constructions such as *half of the*. Overall, then, it appears that membership in TSL is a necessary property of monomorphemic quantifiers, and I conjecture that it may even be a sufficient property when combined with a few additional restrictions.

The paper proceeds as follows: I begin with a brief introduction to the automata-theoretic view of type $\langle 1,1 \rangle$ quantifiers in Sec. 2 and prove that the quantifiers *every* and *no* are much simpler than *an even number of* as only the former are strictly local. Section 3 follows this up with an intuitive discussion of TSL and shows that with the exception of *most* and *half*, monomorphemic quantifiers have TSL string languages. I then argue in Sec. 4.1 that *most* and *half* can be excluded from the class of monomorphemic quantifiers on principled grounds, so that the latter turns out to be properly subsumed by the class of all TSL quantifiers. In an attempt to arrive at an even tighter characterization, I demonstrate in Sec. 4.2 that a handful of very natural restrictions on TSL are sufficient to limit it to exactly the class of attested monomorphemic quantifiers. A formal proof is provided in Sec. 4.3. A few remaining loose ends are discussed in Sec. 5, including the status of *only* (5.1) and proportional *many* (5.2) as well as implications for the learnability of generalized quantifiers (5.3).

## 2 Generalized Quantifiers as String Languages

In order to understand the relation between monomorphemic quantifiers and TSL string languages, the reader first needs to know how generalized quantifiers can be identified with specific string languages (2.1) and how string languages may be classified according to their complexity. For the sake of exposition I focus on only two language classes in this section, the strictly local languages and what I call the *refined strictly local languages*, which are equivalent to the more familiar regular languages (2.2). This bifurcation will be sufficient to demonstrate important complexity differences between quantifiers that so far have all been conflated as regular in the semantic automata tradition (2.3). But the distinction also greatly simplifies the discussion of TSL as an extension of strict locality in Sec. 3.

### 2.1 From Quantifiers to Quantifier Languages

A generalized quantifier 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 generalized quantifier 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).

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, $every(A,B)$ 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 $every(cat, mammal)$ holds. On the other hand, *no cat is a mammal* is false because $no(A,B)$ 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 in terms of which members of $A$ are members of $B$. This is the key to studying them in terms of automata and 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.

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 $e := bac$ of $A$ would be rewritten as 010 by $f_B^A$. For a more concrete example, consider $f_B^A$ with $A$ the set of all males and $B$ the set of all US presidents up to and including 2017. 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 $B$ is the set of all humans, on the other hand, $f$ produces a string with approximately 7 billion symbols, about half of which are 0. The order of 0s and 1s in this string depends on the enumeration of humans and is thus arbitrary.

For any two (countably infinite) sets $A$ and $B$, then, their relation is represented by a (usually non-unique and possibly infinite) string over 0 and 1. I will call this a *binary string over A and B*. A generalized quantifier $Q$, in turn, is equivalent to a language $L$ 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, and $E(A)$ the smallest set of (possibly infinite) strings s over A such that every element of A occurs exactly once in s. The total function $f_B^A$ maps each enumeration $e \in E(A)$ to*

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 $\omega$-automata (Perrin & 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.

*a (possibly infinite) string of 0s and 1s:*

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

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

**Definition 2** (Quantifier Language)**.** *Let $Q$ be a type $\langle 1,1 \rangle$ quantifier. Then its* quantifier language $L(Q)$ *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 $s \in L(Q)$ iff $Q(A,B)$ is true.*

Consider once more the case of *every*. Since *every*$(A,B)$ is true iff $A \subseteq B$, a binary string of $A$ under $B$ only contains 1s whenever *every*$(A,B)$ holds. Conversely, if *every*$(A,B)$ is false there must be at least one $a \in A$ that is not contained in $B$, which implies that every binary string of $A$ under $B$ contains at least one 0. So a binary string is in $L(\textit{every})$ iff it only contains 1s. In other words, $L(\textit{every})$ is the set of all strings over the alphabet $\{1\}$, more succinctly denoted as $1^*$.[2] The same line of reasoning shows that $L(\textit{no})$ contains all and only those strings that contain no 1 (i.e. $0^*$), while $L(\textit{some})$ consists of exactly those strings with at least one 1.

Using $|s|_0$ and $|s|_1$ to denote the number of zeros and ones in a string $s$, respectively, we can succinctly describe these languages:

| Quantifier | Language |
|---|---|
| every | $\lvert s\rvert_0 = 0$ |
| no | $\lvert s\rvert_1 = 0$ |
| some | $\lvert s\rvert_1 \geq 1$ |

Note that these — and in fact all — quantifier languages are closed under permutation. This follows immediately from the definitions above: a binary string of $A$ under $B$ is built from any arbitrary enumeration of $e$ of $A$, and the quantifier language of a quantifier $Q$ has to contain every binary string of $A$ under $B$ if $Q(A,B)$ holds. Permutation-closure will greatly simplify the discussion in Sec. 4.2 as to whether every TSL quantifier language is instantiated by a monomorphemic quantifier in some language.

Numerals are also easily described in terms of the required number of 0s and 1s. However, the string languages differ depending on whether the intended reading establishes a lower bound, an upper bound, or an exact bound on numerosity.

2 The $^*$-operator usually refers to the *Kleene closure*, which consists of all finite strings over a given set. In this paper, I generalize this notation to also include all countably infinite strings.

| Quantifier | Language |
|---|---|
| at least $n$ | $\|s\|_1 \geq n$ |
| at most $n$ | $\|s\|_1 \leq n$ |
| exactly $n$ | $\|s\|_1 = n$ |

Other quantifiers have string languages for which the conditions are slightly trickier to state.[3]

| Quantifier | Language |
|---|---|
| between $m$ and $n$ | $m \leq \|s\|_1 \leq n$ |
| an even number | $\|s\|_1 \mod 2 = 0$ |
| half | $\|s\|_1 = \|s\|_0$ |
| most | $\|s\|_1 > \|s\|_0$ |
| at least one third | $3\|s\|_1 \geq \|s\|_0 + \|s\|_1$ |

The fact that some quantifiers 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.

### 2.2 Two Formal Language Classes for Quantifiers

It is a well-known fact of generalized quantifier theory that proportional quantifiers such as *half*, *most*, and *at least one third* are more complicated than the other quantifiers discussed in the previous section. To establish the complexity split, it suffices to show that the latter are definable in first-order logic whereas the former are not. Formal language theory highlights the very same split, but couches it in different terms: the non-proportional quantifiers like *every*, *no* and *at least 5* have regular quantifier languages, the proportional quantifiers do not.

Before we take a closer look at what it means for a language to be regular, it is worth mentioning that the two views on the complexity split are not equivalent. Büchi (1960) proved that a string language is regular iff it is definable in monadic second-order logic, a proper extension of first-order logic. The logical characterization thus gives us a tighter upper bound on the complexity of non-proportional quantifiers,

3 The definitions in the third table may be inadequate for statements about infinite sets, e.g. *Most natural numbers are not prime*. Since the set of natural numbers and the set of prime numbers are both countably infinite, the sentence would be false under the provided definition of *most*. Instead, the sense seems to make a claim about the low likelihood that any random sample of natural numbers will mostly consist of prime numbers. I do not explore this issue any further because it it arise only with quantifiers that will turn out not to be subregular even when only finite sets are considered.

whereas formal language theory pinpoints the lower bound of proportional quantifiers more accurately:

| | | | | |
|---|---|---|---|---|
| non-proportional quantifiers | $<$ | first-order logic | $<$ | monadic second-order logic |
| monadic second-order logic | $=$ | regular languages | $<$ | proportional quantifiers |

The correspondence between monadic second-order logic and regular languages is not a fluke. Similar connections to logic exist for other formal language classes and will be described in detail where appropriate.

Given the prominent status of regular languages in computer science, it is not surprising that this class was quickly singled out as a major dividing line for the complexity of quantifiers. But it is actually ill-suited to this purpose — despite their low rank in the Chomsky hierarchy, regular languages are very expressive. Not only are all the non-proportional quantifiers discussed so far regular, many unnatural quantifiers are too. For example, the quantifier *less than four or a multiple of three* has a regular quantifier language — this claim is not hard to establish but requires some additional computational background.

Usually one would prove the regularity of a quantifier language by presenting a finite-state automaton that recognize this language. But I will instead use *refined strictly local grammars* to characterize regular languages because they are more closely aligned with the TSL formalism that will dominate the rest of the paper (depending on the reader's computational background, refined strictly local grammars may also be more intuitive than automata).

As is signaled prominently by their name, refined strictly local grammars are an extension of *strictly local grammars*. A strictly local grammar is a finite set of $n$-grams. Each $n$-gram represents a forbidden substring. The language generated by a strictly local grammar contains all strings, and only those, that do not contain any forbidden substrings. In linguistic parlance, strictly local grammars are a collection of inviolable, locally bounded markedness constraints.

**Definition 3** ($n$-gram)**.** *An* $n$-gram *over alphabet* $\Sigma$ *is an element of* $\Sigma^n$*, i.e. a string over* $\Sigma$ *of length n. Given a string s over* $\Sigma$*, its set of n-grams contains all substrings of s of length n and is denoted* $n$-grams$(s)$*. For every such s, its* $n$-augmented *counterpart* $\hat{s}_n := \rtimes^n \cdot s \cdot \ltimes^n$ *is obtained by adding n instances of the left and right edge markers* $\rtimes$ *and* $\ltimes$ *to s. By assumption,* $\rtimes$ *and* $\ltimes$ *are distinguished symbols not contained in* $\Sigma$*.*

**Definition 4.** *A* strictly $n$-local grammar *G over* $\Sigma$ *is a finite (and possibly empty) set of n-grams over the alphabet* $\Sigma \cup \{\rtimes, \ltimes\}$*. The language of G is given by* $L(G) := \{s \mid n\text{-grams}(\hat{s}_{n-1}) \cap G = \emptyset\}$*. A string language L is* strictly $n$-local *only if there is*

*some strictly n-local grammar G with $L(G) = L$. We say that L is* strictly local *iff it is strictly n-local for some $n \geq 0$.*

A brief remark is in order regarding the presentation of $n$-grams. According to the definition, an $n$-gram is always exactly of length $n$, no longer and no shorter. Strings are padded out with enough left and right edge markers to ensure that their length is at least $n$. Often, though, it is convenient to shorten $n$-grams in the definition of a strictly $n$-local grammar. For example, if a strictly 3-local grammar blocks the empty string, it has to contain at least one of the trigrams $\rtimes \rtimes \ltimes$ and $\rtimes \ltimes \ltimes$. Instead, one can just use the bigram $\rtimes \ltimes$ given the implicit understanding that it must be padded out to an equivalent trigram. I adopt this simpler notation throughout the paper, which is of particular importance for Definition 6 in Sec. 4.2.

Let us now try to apply the notions above to some quantifier languages. Rather than look immediately at the fairly complex case of *less than four or a multiple of three*, it is prudent to start with some simpler, non-semantic examples from formal language theory. Consider the language $(ab)^*$, which contains all strings that can be built from zero or more iterations of *ab*: the empty string $\varepsilon$, *ab*, *abab*, and so on. Among the illicit strings we find *a*, *b*, *ba*, *aa*, *bb*, *aba*, *ababbab*, and many more. Using $\rtimes$ and $\ltimes$ to mark the left and right edge of a string, respectively, we can characterize this language with a handful of markedness constraints.

| **Constraint** | **Forbidden $n$-gram** |
|---|---|
| Don't start with *b* | $\rtimes b$ |
| Don't end with *a* | $a \ltimes$ |
| *a* never follows *a* | *aa* |
| *b* never follows *b* | *bb* |

No member of $(ab)^*$ violates any of these constraints, and every string not belonging to $(ab)^*$ violates at least one. Therefore the set $\{\rtimes b, a\ltimes, aa, bb\}$ forms a strictly local grammar $G$ that generates the language $(ab)^*$. Since $G$ contains only bigrams we also say that $G$ is *strictly 2-local*.

Now consider the language $(aa)^*$, which is indirectly related to the quantifier *an even number of*. This language contains the empty string $\varepsilon$, *aa*, *aaaaa*, *aaaaaa*, and so on. In other words, it contains all strings of an even length, and only those. The language is not generated by any strictly local grammar. A strictly local grammar must only contain a finite number of $n$-grams, for some fixed $n$. Now pick some string $s$ in $(aa)^*$ whose length $l$ is greater than $n$. Since $s$ is in the language, the result of adding one more *a* to $s$ is an ill-formed string $s'$. Yet $s$ and $s'$ contain exactly the

same $n$-grams: $\rtimes a^{n-1}$, $a^n$, and $a^{n-1}\ltimes$.[4] A grammar that does not generate $s'$ must ban one of these $n$-grams. But then this grammar does not generate $s$, either. Since $n$ is arbitrary, this problem arises with every strictly local grammar. It follows that no strictly local grammar generates $(aa)^*$.

Refined strictly local grammars, on the other hand, can generate $(aa)^*$. In a refined strictly local grammar, strings are also annotated with a hidden alphabet (which serves the same function as states in finite state automata). The $n$-grams of a refined strictly local grammar reference this hidden alphabet to disallow certain sequences. For $(aa)^*$, the grammar lists the following refined bigrams as forbidden:

$$\frac{e}{\rtimes a} \quad \frac{o}{a\ltimes} \quad \frac{ee}{aa} \quad \frac{oo}{aa}$$

Intuitively, $e$ and $o$ are hidden symbols that keep track of whether the string contains an even or an odd number of $a$s up to and including the current position. The forbidden refined bigrams ensure that a string must start with the hidden symbol $o$ and end with $e$, a requirement that can only be satisfied by strings of even length.

Let us first consider the well-formed string $aa$. There are four possible ways of annotating this string with a hidden alphabet:

$$\frac{oe}{aa} \quad \frac{oo}{aa} \quad \frac{ee}{aa} \quad \frac{eo}{aa}$$

Only the first annotation scheme is licit because it does not contain any offending refined bigram. The second string has $o$ followed by $o$; in string three, $e$ follows $e$; and string four ends in $o$. All of these configurations are blocked by the grammar. But even though only the first string constitutes a successful annotation, this is sufficient to classify the string as well-formed. As long as we can find at least one valid assignment of hidden symbols, the string will be generated by the grammar.

Now contrast this with the illicit string $aaa$. There are eight possible annotations, but every single one of them necessarily involves an illicit refined bigram.

$$\frac{ooo}{aaa} \quad \frac{ooe}{aaa} \quad \frac{oeo}{aaa} \quad \frac{oee}{aaa} \quad \frac{eoo}{aaa} \quad \frac{eoe}{aaa} \quad \frac{eeo}{aaa} \quad \frac{eee}{aaa}$$

Since every logically possible annotation of $aaa$ is blocked, the string is not generated by the refined strictly local grammar.

I refrain from defining refined strictly local grammars in formal terms because the complexity of the definition greatly outweighs the relevance of that class for this paper. The important insight is that there are two classes of grammars, strictly local

4 Strictly speaking the set of $n$-grams consists of all strings of the form $\rtimes^u a^v \ltimes^w$, where $u, v, w \geq 0$ and $u+v+w = n$. This complication does not affect the validity of the argument but reduces the clarity of exposition.

grammars and refined strictly local grammars. The latter are exactly as powerful as finite-state automata and are thus capable of generating all regular languages (and only those). Strictly local grammars are much weaker and can only generate strictly local languages, which includes $(ab)^*$ but not $(aa)^*$. This split between strictly local grammars on the one hand and refined strictly local on the other allows for a more fine-grained characterization of the complexity of string languages. In the remainder of this section, I demonstrate the usefulness of said split for quantifier languages. Section 3 then argues that the distinctions are still too coarse and TSL is needed as an intermediate class between strictly local and refined strictly local.

### 2.3 Examples of Strictly Local and Regular Quantifier Languages

With the basics of refined strictly local grammars in place, let us return to quantifier languages and how many of them can be shown to be regular. The grammar for $(aa)^*$ from the previous section is easily modified so that it generates the quantifier language of *an even number of*. We now use *e* and *o* to keep track of the number of 1s rather than the number of symbols. The forbidden refined *n*-grams are a follows:

$$\begin{array}{cccc} \overline{\overset{o}{\rtimes 0}} & \overline{\overset{e}{\rtimes 1}} & \overline{\overset{o}{0 \ltimes}} & \overline{\overset{o}{1 \ltimes}} \\ \overline{\overset{eo}{00}} & \overline{\overset{oe}{00}} & \overline{\overset{ee}{01}} & \overline{\overset{oo}{01}} \\ \overline{\overset{eo}{10}} & \overline{\overset{oe}{10}} & \overline{\overset{ee}{11}} & \overline{\overset{oo}{11}} \end{array}$$

The *n*-grams ensure I) that the alternation between *e* and *o* proceeds as desired depending on whether the current symbol is 0 or 1, and II) that only strings with *e* as the final annotation are generated. A few illustrative examples of well-formed and ill-formed binary strings are depicted below.

$$\overline{\overset{eoee}{0110}} \quad \overline{\overset{eoeooe}{011101}} \quad *\overline{\overset{eoeo}{0111}} \quad *\overline{\overset{eoee}{0111}}$$

Returning to our initial example of *less than four or a multiple of three*, we can now show its quantifier language to be regular, too. However, the grammar is much more complicated than anything we have seen so far (the grammar given here treats *less than* as denoting *strictly less than*, which slightly simplifies things). The hidden alphabet consists of 0, 1, 2, $3_0$, $3_1$, and $3_2$. This is used to keep track of the total number of 1s up to a threshold of 3, at which point counting switches from absolute values to *modulo 3*. The grammar also contains many more forbidden bigrams than any of the previous ones, and for this reason I use several notational shorthands. First, $\neg\sigma$ is a stand-in for any hidden symbol other than $\sigma$, and $\sigma\ \neg\sigma$

denotes any sequence where the second hidden symbol does not match the first one. Furthermore, $\sigma\ \neg\sigma + 1$ represents any sequence of hidden symbols where the second symbol is not the successor of the first one, according to the following order: $0 < 1 < 2 < 3_0 < 3_1 < 3_2 < 3_0 < 3_1 < 3_2 < 3_0 \cdots$.

$$\frac{\neg 0}{\rtimes 0} \quad \frac{\neg 1}{\rtimes 1} \quad \frac{3_1}{0\ \ltimes} \quad \frac{3_1}{1\ \ltimes} \quad \frac{3_2}{0\ \ltimes} \quad \frac{3_2}{1\ \ltimes}$$

$$\frac{\sigma\ \neg\sigma}{0\ \ 0} \quad \frac{\sigma\ \neg\sigma}{1\ \ 0} \quad \frac{\sigma\ \neg\sigma + 1}{0\ \ \ 1} \quad \frac{\sigma\ \neg\sigma + 1}{1\ \ \ 1}$$

Without the notational shorthands, this grammars has 134 refined bigrams. Annotated examples of well-formed strings are shown below. The reader is invited to verify that ill-formed strings like 11111 or 111011101 cannot be generated.

$$\frac{0}{0} \quad \frac{0\ 1\ 2\ 3_0}{0\ 1\ 1\ 1} \quad \frac{0\ 1\ 2\ 3_0\ 3_0\ 3_1\ 3_2\ 3_0}{0\ 1\ 1\ 1\ \ 0\ \ 1\ \ 1\ \ 1}$$

Let us consider one more example, the quantifier language for *some*. As the complex *less than four or a multipe of three* has a regular quantifier language, it is not surprising that *some* does, too. However, the grammar is much simpler and forbids only a few refined $n$-grams.

$$\frac{1}{\rtimes 0} \quad \frac{0}{\rtimes 1} \quad \frac{0}{0\ltimes}$$

$$\frac{01}{00} \quad \frac{10}{00} \quad \frac{00}{01} \quad \frac{10}{10} \quad \frac{10}{11}$$

This grammar uses the hidden alphabet to keep track of whether at least one 1 occurs in the string, which is sufficient for *some*.

Intuitively, it seems odd to say that *some* and *less than four or a multiple of three* belong to the same class, whereas *most* or *more than a third* are more complex because their quantifier languages are not regular. After all, *some* is not just simpler than *most* and *more than a third*, it also appears much simpler than *less than four or a multiple of three*. One reply to this observation is that *some* is indeed simpler because its refined strictly local grammar is much smaller: instead of 134 bigrams, it has only 8.

But such succinctness claims do not describe intrinsic properties of quantifier languages, they are dependent on the formal machinery that generates these languages. Succinctness can vary a lot between different types of machinery — for instance, formulas of monadic second-order logic can be much more succinct than finite state automata, which in turn are more succinct than refined strictly local grammars. So the difference we observe between *some* and *less than four or a multiple of*

*three* may be much less pronounced with another description device. Suppose for the sake of argument that we use monadic second-order logic with dedicated logical operators $<_n$ and $\times_n$ for *less than n* and *multiple of n*, respectively, but without the familiar first-order quantification. Then the quantifier language for *less than four or a multiple of three* would be defined by $<_4 (1) \vee \times_3(1)$, whereas $L(\textit{some})$ would correspond to $\neg <_1 (1)$. Both descriptions are extremely succinct. In fact, the formula for *some* ($\neg <_1 (1)$) would be more complex than that for *a multiple of three* ($\times_3(1)$). Artificial as the example may be, it illustrates that one should be careful with succinctness claims in the absence of a firm, independently justified basis for comparison.

It would be much more appealing if the intuitive contrast between *some* and *less than four or a multiple of three* could be explained in the same manner as the difference between *some* and *most*: the latter belongs to a more complex language class. In fact, this is already possible for *every* and *no*. Their quantifier languages aren't just regular, they even belong to the much weaker class of strictly local languages. For *every*, no binary string may contain any instance of 0. For *no*, on the other hand, 1 must not occur in any string. This is expressed by the strictly 1-local grammars $\{0\}$ and $\{1\}$, respectively. These are the simplest non-trivial grammars over the alphabet $\{0,1\}$.[5] That *every* and *no* are strictly local conforms with the intuition that they both are much simpler than *less than four or a multiple of three*, which is regular but not strictly local.

It would be desirable to extend this kind of argument to *some*, but unfortunately its quantifier language is not strictly local. Remember that $L(\textit{some})$ consists of all strings with at least one 1. The requirement for the presence of 1 cannot be translated into a local ban against certain substrings, which is a prerequisite for describing it with a strictly local grammar.

A formal proof is straight-forward. Consider any arbitrary string $0^n10^n \in L(\textit{some})$ and contrast that with $0^n \notin L(\textit{some})$. All the $n$-grams of the latter are also part of the former, so a strictly $n$-local grammar cannot block $0^n$ without also incorrectly blocking $0^n10^n$. Since $n$ is arbitrary, $L(\textit{some})$ cannot be strictly local.

We see, then, that replacing the monolithic class of regular languages by the hierarchy *strictly local* < *regular* is not enough to capture all apparent complexity contrasts. While we do get a neat split with *empty* and *no* as strictly local and *an even number of* or *less than four or a multiple of three* as regular, there is insufficient room to properly accommodate quantifiers like *some*. In the next section, we will see that there is a much more articulate hierarchy of regular string languages, with strictly local at the bottom and regular at the top. Many natural language quantifiers

5 Only the empty grammar is even simpler, but this grammar generates every possible binary string. In other words, the empty grammar generates $\{0,1\}^*$, the quantifier language of the tautological quantifier $\top$, where $\top(A,B)$ is true for every choice of $A$ and $B$.

can be found close to the bottom of the hierarchy, and in particular, all quantifiers that can function as determiners and are widely believed to be monomorphemic are captured by a minor extension of the strictly local grammars: TSL.

## 3 Subregular Quantifier Languages: Tier-Based Strict Locality

After their introduction in Chomsky (1956), it quickly became apparent that regular languages are not the weakest conceivable class of string languages. The first results along these lines are Schützenberger (1965) and McNaughton & Papert (1971), which gave rise to an active research enterprise (see Pin 1997 for an extensive survey). The result was the *subregular hierarchy*, which includes many classes besides the strictly local languages we already encountered in the previous section. For a long time the implications of the subregular hierarchy remained largely unexplored by theoretical and computational linguists alike. But in recent years several researchers have developed an interest in subregular languages with respect to animal communication (Pullum & Rogers 2006, Rogers & Pullum 2011) and phonology (Heinz 2007, 2009, 2010a,b, 2014, 2015, Graf 2010, 2017, McMullin & Hansson 2015, McMullin 2016; see also Bird 1995 and Potts & Pullum 2002 for early precursors). This perspective proved so fruitful that it is now also being applied to morphology (Aksënova et al. 2016, Chandlee 2016) and syntax (Graf & Heinz 2015). The new-found interest in subregular languages prompted several additions to the subregular hierarchy, an up-to-date version of which is shown in Fig. 1.

Given the expansive nature of the subregular hierarchy, it is not feasible to discuss all its language classes here (the interested reader is referred to Rogers & Pullum 2011, Heinz 2015, and Graf 2017). Instead, I will focus on one particular class that has already been found to play a central role in phonology, morphology, and syntax: *tier-based strictly local* (TSL; Heinz et al. 2011). Like the refined strictly local grammars, TSL grammars are a more powerful variant of strictly local grammars. However, they are still much weaker than refined strictly local grammars. Whereas those use an additional hidden alphabet for string annotations, a TSL grammar combines a strictly local grammar with a mechanism for masking specific non-adjacent symbols in the string adjacent via a tier.

**Definition 5** (Tier-Based Strictly Local)**.** *A* tier-based strictly local *grammar $G$ is a pair $\langle S, T \rangle$ where $S$ is a strictly local grammar over a fixed alphabet $\Sigma$ and $T \subseteq \Sigma$ is a tier alphabet. We say that $G$ is* tier-based strictly $n$-local *(TSL-$n$) iff $S$ is strictly $n$-local (SL-$n$).*

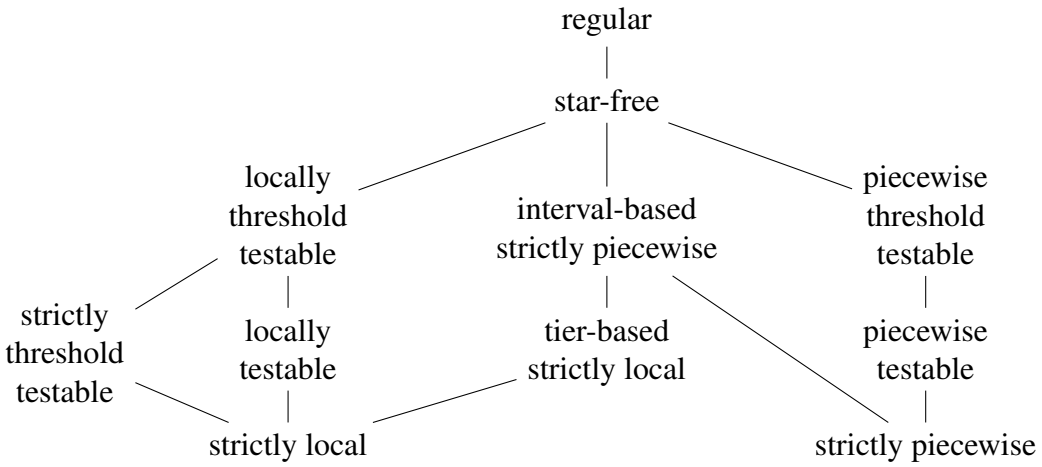



**Figure 1** The subregular as developed in Schützenberger (1965), McNaughton & Papert (1971), McNaugthon (1974), Simon (1975), and Ruiz et al. (1998), a.o., with recent extensions for computational phonology (Rogers et al. 2010, Heinz et al. 2011, Graf 2017).

*Given a string $s$ over $\Sigma$, its $T$-tier $T(s)$ is given by:*

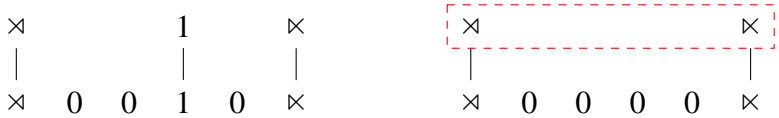
$$T(s) := \begin{cases} T(u) \cdot T(s') & \text{if } u \in \Sigma \text{ and } s = u \cdot s' \\ s & \text{if } s \in T \\ \varepsilon & \text{otherwise} \end{cases}$$

*The language generated by $G$ is $L(G) := \{s \in \Sigma^* \mid T(s) \in L(S)\}$, and a string set is tier-based strictly local iff there is a tier-based strictly local grammar that generates it.*

Intuitively, a TSL grammar "masks out" all symbols in the string that do not belong to the specified tier alphabet. If the remainder is well-formed with respect to a given strictly local grammar, the whole string is considered well-formed. While this ability to ignore certain parts of the string serves many purposes in phonology, the main payoff for generalized quantifiers is the ability to do some limited counting.

Consider once more the quantifier language for *some*. As previously explained in Sec. 2.3, this language is regular but not strictly local. However, $L(\textit{some})$ is TSL — the requirement that we have at least one 1 is satisfied iff masking out all 0s does not produce the empty string. This translates into the TSL grammar $G := \langle \{\rtimes \ltimes\}, \{1\} \rangle$. In phonological parlance, the grammar projects a 1-tier and

requires said tier to be non-empty. The phonological conceptualization offers an intuitive way of depicting the relevant contrast:




The counting power of TSL grammars can be extended from *some* to any numeral. For instance, replacing the bigram $\rtimes\ltimes$ by the 4-gram 1111 results in blocking all strings with four or more 1s. In other words, this new grammar defines the quantifier language for *at most 3*.


For *at least 3*, one has to forbid tiers with less than three 1s using the following set of $n$-grams: $\{\rtimes\ltimes, \rtimes 1\ltimes, \rtimes 11\ltimes\}$.


Adding the 4-gram 1111 to this set yields a TSL grammar that generates $L(\textit{exactly 3})$. If instead of 1111 we had added the 5-gram 11111, the resulting language would have been $L(\textit{between 3 and 4})$.


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

Note that all these quantifiers are modeled by TSL grammars with tier alphabet $\{1\}$. Instead of this tier alphabet, one could also have $\{0\}$, the empty set, and $\{0, 1\}$. Changing the tier alphabet to 0 allows for the expression of some other generalized quantifiers.

| Grammar | Quantifier |
|---|---|
| $\{\rtimes\ltimes\}$ | not all |
| $\{\rtimes\ltimes, 00\}$ | all but one |
| $\{\rtimes\ltimes, \rtimes 0\ltimes, 000\}$ | all but two |
| $\vdots$ | |

TSL grammars with an empty tier alphabet are useless, as this will always result in an empty tier for every binary string. If the grammar blocks $\rtimes\ltimes$, no binary strings are allowed and we obtain the language $\emptyset = L(\bot)$, where $\bot(A,B)$ is always false irrespective of the choice of $A$ and $B$. If, on the other hand, the grammar does not block $\rtimes\ltimes$ on the tier, the generated language is $\{0,1\}^* = L(\top)$, where $\top(A,B)$ is true for every choice of $A$ and $B$. Finally, a tier alphabet of $\{0,1\}$ renders the tier identical to the binary string it is projected from, and consequently such TSL grammars are equivalent to strictly local grammars because they fail to exploit the power of tier projection. As a welcome side-effect, this immediately establishes that every strictly local quantifier language is also TSL, including $L(\textit{every})$ and $L(\textit{no})$.

Even though TSL accommodates a variety of quantifiers, there are still many generalized quantifiers that are not TSL. Since every TSL language is also regular, 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, whereas projecting both 0 and 1 reduces TSL grammars to strictly local grammars, which we already know to be insufficient for *an even number*. 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 as we saw during the discussion of the string language $(aa)^*$ in Sec. 2.2, strictly local grammars cannot distinguish strings of even length from those of odd length. Since every TSL grammar uses a strictly local grammar to distinguish well-formed from ill-formed tiers, the quantifier language for *an even number* simply cannot be TSL.

Putting all our findings together, we arrive at a more fine-grained view of quantifier complexity, displayed in Tab. 1. The table makes it very clear that TSL encompasses most of the quantifiers that linguists are inclined to consider natural (as opposed to more mathematical quantifiers like *an even number*). In the next section, I argue that this is not a coincidence and that TSL plays a central role for natural language quantifiers.

| strictly local | < | TSL | < | regular | < | context-free |
|---|---|---|---|---|---|---|
| *every* | | *some* | | *an even number* | | *most* |
| *no* | | *not all* | | | | *half* |
| | | *at most n* | | | | *at least one third* |
| | | *at least n* | | | | |
| | | *exactly n* | | | | |

**Table 1** Classification of natural language quantifiers by the complexity of their quantifier languages

## 4 The TSL Nature of Monomorphemic Quantifiers

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. In this section I argue that the subregular hierarchy not only provides a more fine-grained classification of generalized quantifiers, it also picks out natural classes. All monomorphemic quantifiers are TSL once one excludes *most* and *half* (4.1), so TSL is an upper bound on the complexity of non-proportional monomorphemic quantifiers. Since there are independent reasons not to consider *most* and *half* as instances of monomorphemic quantifiers, one may conjecture that all monomorphemic quantifiers in natural languages are TSL. In fact, a few natural restrictions on TSL allow it to pick out exactly the class of all monomorphemic quantifiers (4.2) *modulo* existential import.

### 4.1 All Monomorphemic Quantifiers are TSL

Based on articles from the *Handbook of Quantifiers in Natural Language* (Keenan & Paperno 2012), Paperno (2011) presents a list of monomorphemic quantifiers sampled from 13 languages: Adyghe, Basque, Garifuna, German, Hebrew, Hungarian, Italian, Malagasy, Mandarin, Pima, Russian, Telugu, and Western Armenian. With the exception of 'one and a half' in Russian, this list does not contain any quantifiers that aren't also monomorphemic in English. If one puts aside temporal (*always*, *often*) or inherently vague quantifiers (*few*, *several*, *many*), the remaining list consists of *every*/*all*, *no*, *some*/*one*, *most*, *half*, and various numerals.

Except for the proportional quantifiers, all these quantifiers are TSL. While this may seem like pure coincidence, the fact that many patterns in phonology, morphology, and syntax have recently been shown to be TSL suggests that something more profound may be at play.

**Conjecture 1.** *If a natural language contains a monomorphemic quantifier Q, then $L(Q)$ must be TSL.*

Seeing how effortlessly TSL grammars captured a number of quantifiers in the previous section, this conjecture may appear unambitious. But from the perspective of formal language theory, TSL is a very small and weak fragment of the class of regular languages. So the relative ease with which TSL can be applied to natural language quantifiers only goes to show how simple many of them are.

That said, the conjecture is obviously threatened by the existence of at least two apparent counterexamples in the form of *most* and *half*. One could point towards the typological rarity of those quantifiers — in Paperno's (2011) list, Telugu and Russian are the only languages besides English with 'most' (homophonic with 'many'), and monomorphemic 'half', respectively. While this is certainly noteworthy and may be the vantage point for a quantitative analysis correlating quantifier language complexity with typological frequency, stronger arguments exist that preserve the conjecture above as a universal rather than a statistical tendency. The crucial issue is whether *most* and *half* qualify as monomorphemic quantifiers upon closer scrutiny.

Hackl (2009) presents a well-known analysis of *most* as the superlative of *many*. Decomposing *most* in this fashion makes it easier to derive the difference in meaning between *John climbed most of the mountains* and *John climbed the most mountains*. Even though Hackl's proposal has not been widely adopted in the generalized quantifiers community, the two are far from incompatible. It is perfectly reasonable to factor *most* into several components, the interaction of which yields the behavior of the generalized quantifier $most(A,B)$ iff $|A \cap B| > |A - B|$. But crucially this factorization no longer treats *most* as monomorphemic. Adopting Hackl's analysis of *most* thus has no major drawbacks for the enterprise of studying generalized quantifiers yet it explains the otherwise mysterious placement of *most* outside TSL and even the regular languages.

This leaves us with *half*, for which no decompositions analogous to Hackl (2009) have been offered in the literature. The only tangible peculiarity of *half* is that it cannot be used as a determiner in English or Russian, where it always requires genitive marking on the noun, suggesting a different syntactic structure.

(1) a. half *(of the) men
b. polovina človek*(-a)
half man-GEN
c. polčelovek*(-a)
half.man-GEN

So if the term *quantifier* in the conjecture is taken to refer only to quantifiers that function as determiners or numerals, *half* no longer constitutes a counterexample.

Still, handwaving away a semantic counterexample via recourse to syntactic distinctions is unsatisfying. A proper semantic explanation would go much farther in solidifying the empirical validity of the conjecture; hopefully one will be presented in the near future.

The reader may still wonder why the TSL conjecture is appealing enough to warrant such an elaborate defense. After all, the subregular hierarchy in Fig. 1 furnishes numerous classes, so maybe another one provides a better fit for monomorphemic quantifiers. This is partially true. Among the classes that do not subsume TSL, the locally threshold testable (LTT) languages — and only those — have comparable empirical coverage to TSL. All other classes such as strictly piecewise and strictly threshold testable fail even for basic quantifiers, e.g. *some* and *at least n*.

Between TSL and LTT, it is TSL that provides the better fit because it does not overgenerate as much. A set of binary strings is LTT iff it can be defined in first-order logic. For example, the quantifier language for *between 3 and 5* is represented by the following formula:

$$\exists x_1, x_2, x_3[\bigwedge_{1 \leq i \leq 3} 1(x_i) \wedge \bigwedge_{1 \leq i < 3} x_i \neq x_{i+1}] \wedge \neg\big(\exists x_1, \ldots, x_6[\bigwedge_{1 \leq i \leq 6} 1(x_i) \wedge \bigwedge_{1 \leq i < 6} x_i \neq x_{i+1}]\big)$$

But the class of first-order definable quantifiers is clearly too large for natural language. In first-order logic, one could define a quantifier $Q$ such that $Q(A,B)$ iff either $A$ contains exactly two elements or if at least five elements of $A$ belong to $B$, then ten elements of $A$ do not belong to $B$. While such a quantifier can be paraphrased in natural language, it certainly isn't grammaticalized, let alone lexicalized as a monomorphemic word. Not only does TSL avoid such cases of massive overgeneration, it can actually be restricted in a natural manner to generate only quantifier languages of monomorphemic quantifiers.

### 4.2 (Almost) All TSL Quantifiers are Monomorphemic

Table 2 gives an overview of the quantifiers discussed in this paper. The strings of their quantifier languages are described using the $|s|_i$ notation, which denotes the number of occurrences of symbol $i$ in string $s$. The table highlights that even if TSL may be a necessary property for monomorphemic quantifiers (with the possible exception of *most* and *half*), it is not a sufficient one. No language has a monomorphemic equivalent of *not all*, *all but n* or *between m and n*, yet their quantifier languages are TSL. But there are formal aspects of these quantifier languages that set them apart from the rest so that a few natural restrictions on TSL suffice to narrow down the class of definable quantifiers to exactly the monomorphemic ones. The precise characterization depends on what exactly one takes to be the class of possible monomorphemic quantifiers — several options are discussed below in an informal

| Quantifier | String Condition | Complexity | Grammar |
|---|---|---|---|
| *every* | $\|s\|_0 = 0$ | SL-1 | $\{0\}$ ($T := \{0,1\}$) |
| *no* | $\|s\|_1 = 0$ | SL-1 | $\{1\}$ ($T := \{0,1\}$) |
| *some* | $\|s\|_1 \geq 1$ | TSL-2 | $\{\rtimes\ltimes\}$, $T := \{1\}$ |
| *not all* | $\|s\|_0 \geq 1$ | TSL-2 | $\{\rtimes\ltimes\}$, $T := \{0\}$ |
| *(at least) n* | $\|s\|_1 \geq n$ | TSL-$(n+1)$ | $\left\{\rtimes 1^k \ltimes\right\}_{k<n}$, $T := \{1\}$ |
| *(at most) n* | $\|s\|_1 \leq n$ | TSL-$(n+1)$ | $\left\{1^{n+1}\right\}$, $T := \{1\}$ |
| *(exactly) n* | $\|s\|_1 = n$ | TSL-$(n+1)$ | $\left\{1^{n+1}, \rtimes 1^k \ltimes\right\}_{k<n}$, $T := \{1\}$ |
| *between m and n* | $m \leq \|s\|_1 \leq n$ | TSL-$(n+1)$ | $\left\{1^{n+1}, \rtimes 1^k \ltimes\right\}_{k<m}$, $T := \{1\}$ |
| *all but n* | $\|s\|_0 = n$ | TSL-$(n+1)$ | $\left\{0^{n+1}, \rtimes 0^k \ltimes\right\}_{k<n}$, $T := \{0\}$ |
| *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 2** String complexity for some generalized quantifiers in English

fashion, with Sec. 4.3 providing a rigorous summary of the different characterizations. All these characterizations, however, limit the set of possible tier alphabets and require the TSL grammars to satisfy certain closure properties with respect to the $n$-grams they contain.

**Tier alphabet** The first restriction concerns the tier alphabet. Let us temporarily ignore *every* and *no*, which are strictly local and thus do not require any tier at all. Among the remaining TSL quantifiers, *not all* and *all but n* stand out because they require the tier alphabet to be $\{0\}$ rather than $\{1\}$ (recall the discussion towards the end of Sec. 3). If for some reason $\{0\}$ is not a licit tier alphabet, then these quantifiers are no longer expected to have monomorphemic instantiations in any natural languages. From the perspective of formal language theory, this restriction is extremely *ad hoc* since 0 and 1 are just arbitrary symbols without special status. In the case of binary string languages, however, the symbols do carry meaning: they encode $B$-membership and $B$-non-membership, respectively. A tier alphabet of $\{0\}$ gives rise to a quantifier that prioritizes non-membership over membership. This seems rather counterintuitive, and one might speculate that certain general preferences of human cognition adjudicate against such an inverted notion of prominence. A tier alphabet of $\{0,1\}$, on the other hand, is tantamount to having no tier at all and thus conferring no special status to either 0 or 1. It appears, then, that monomorphemic quantifiers across languages are such that either 0 and 1 have equal status or 1 is more prominent than 0 (in other words, the tier alphabet must always contain 1).

**Excluding *between* and *exactly*** Ruling out *between m and n* as a potential monomorphemic quantifier forces us to take less appealing steps. Close inspection of Tab. 2 reveals that it is impossible to block *between m and n* without also ruling out *exactly n*. After all, *exactly n* is logically equivalent to *between n and n*. However, removal of *exactly n* from the list of potential monomorphemic quantifiers is not necessarily unwelcome. There is plenty of semantic and psycholinguistic evidence that the semantic denotation of a numeral like *four* is *at least four*, with the exact reading brought about by pragmatic strengthening. If we assume that *exactly n* is not a viable denotation for numerals, then we can rule out the remaining undesired quantifiers with a single restriction: the set of illicit tier $n$-grams must be of the form $\left\{ \rtimes 1^k \ltimes \right\}_{k<n}$, $n > 0$. This also has the welcome benefit of ruling out monomorphemic quantifiers with the meaning of "strictly less or strictly more than four". Nonetheless future work will have to further motivate this restriction if it is to prove more than a technical trick.

**Including vague quantifiers** There is also an alternative way of dealing with *between m and n*. The implicit assumption so far has been that this quantifier has no monomorphemic instantiation. But vague quantifiers such as *few*, *several*, and *many* can be viewed as lexicalized versions of *between m and n* where the values of *m* and *n* are supplied by the context. Similarly, *exactly n* may be an instance of *between m and n* where the context requires $m = n$. Or maybe semantics furnishes *exactly n* as a reading of numerals without the help of pragmatics — any solution works for our purposes as long as both *between m and n* and *exactly n* are either included in or excluded from the class of monomorphemic quantifiers.

**Excluding disjunctive quantifiers** If *between m and n* and *exactly n* are included in the set of monomorphemic quantifiers, a dedicated stipulation is required to exclude "strictly less or strictly more than four" as a possible denotation for monomorphemic quantifiers: if $\rtimes 1^n \ltimes$ is illicit, then so is $\rtimes 1^{n-1} \ltimes$. This amounts to a kind of downwards closure requirement. Note that an analogues upwards closure requirement cannot be formulated. Such a condition would require a grammar to contain $\rtimes 1^n \ltimes$ whenever it contains $\rtimes 1^{n-1} \ltimes$. A TSL grammar can only contain a finite number of $n$-grams, whereas this closure property necessarily result in an infinite set. So no TSL grammar can satisfy upwards closure. On the other hand, an upwards closure requirement with respect to $n$-grams of the form $1^n$ is redundant: any string with $1^{n+1}$ as a substring also has $1^n$ as a substring and thus will be blocked by a grammar with $n$-gram $1^n$. In a certain sense, then, TSL grammars already incorporate a kind of upwards closure requirement, and downwards closure is simply its symmetric counterpart.

**Excluding domain requirements** There is one more case of overgeneration, which arises when the tier alphabet is $\{0,1\}$, which is necessary for *every* and *no*. Their respective quantifier languages are generated by the grammars $\{0\}$ and $\{1\}$ over such tiers. Nothing so far prevents us to move from unigrams to larger $n$-grams, in fact this is necessary to express the counting quantifiers *at least n*, *at most n*, and *exactly n* as TSL grammars with the tier alphabet $\{1\}$. But with a tier alphabet of $\{0,1\}$, moving beyond unigrams allows for highly unnatural quantifier languages.

Consider the grammar $G := \{00,01,10,11\}$, which generates the set of all strings whose length is at most 1. This set is permutation closed and thus a quantifier language. The quantifier it encodes is *at most one thing is an A*, or more formally: $Q(A,B)$ iff $|A| < 2$. This is not a plausible meaning for a type $\langle 1,1\rangle$ quantifier, let alone a monomorphemic one. If one adds the bigram $\rtimes\ltimes$ to $G$, one instead obtains the quantifier $Q(A,B)$ iff $|A| = 1$. And the grammar $\{\rtimes\ltimes, \rtimes 1\ltimes\}$ defines a quantifier that requires $A$ to contain more than two elements or at least one element that is not a $B$. None of these may be part of a formal characterization of possible meanings of monomorphemic quantifiers.

The central problem is that the ability to define minimum and maximum lengths for strings is required for numerals, where the tier alphabet is $\{0,1\}$, but when the tier alphabet is $\{0,1\}$ this instead enforces size requirements on the domain $A$. Admittedly this has the advantage that it becomes possible to capture existential import by ruling out the empty string: the variant of *every* that presupposes a non-empty domain of quantification is captured by the grammar $\{\rtimes\ltimes, 0\}$. But as we just saw, this is a very limited use of the power that arbitrary $n$-grams afford. I do not have an insightful explanation as to why counting is very productive over $\{1\}$-tiers but not over tiers with alphabet $\{0,1\}$. At this point, this must be ruled out by stipulation: if $T := \{0,1\}$, then $G$ must be a unigram grammar. With this severe limitation, even existential import becomes impossible to express, so one might relax it a bit to also allow $\rtimes\ltimes$ as the only non-unigram.

While this stipulation is hardly insightful, it at least reduces the unavailability of more complex strictly local quantifier languages to a well-established property of natural language quantifiers: they do not pay attention to the size of the domain.

### 4.3 Formal Summary

The previous two sections have introduced a lot of conceivable scenarios for how the class of monomormphemic quantifiers may be constituted and how TSL can be restricted to generate only quantifier languages for members of this class. For the sake of explicitness, let us once more summarize these conditions and how they derive the posited range(s) of monomorphemic quantifiers.

**Definition 6.** *A TSL grammar $G := \langle S, T \rangle$ over alphabet $\{0,1\}$ is*

- non-trivial *iff $G$ does not generate the empty language $\emptyset$ or $\{0,1\}^*$;*
- truth affine *iff $\{1\} \subseteq T$;*
- domain agnostic *iff $T := \{0,1\}$ implies $S \subseteq \{0,1,\rtimes,\ltimes\}$;*
- downward closed *iff $\rtimes 1^k \ltimes \in S$ implies $\rtimes 1^{k-1} \ltimes \in S$.*

**Theorem 1.** *Suppose $L$ is permutation-closed and generated by a non-trivial, truth affine, domain agnostic, downward closed TSL grammar over $\{0,1\}$. Then $L$ is the quantifier language for one of the following:*

- every,
- no,
- some,
- at least *n,*
- at most *n,*
- exactly *n,*
- between *m* and *n.*

*Proof.* The theorem is established by enumeration all possible types of grammars $G := \langle S, T \rangle$ that meet the required conditions. We separate the cases by the choice of $T$. We can immediately exclude $T := \{\}$ and $T := \{0\}$ because $G$ is truth affine.

*Case 1* $T := \{0,1\}$. In this case $G$ behaves exactly like its strictly local grammar $S$. Since $S$ is domain agnostic, it may only contain unigrams. But since $S$ is non-trivial, it cannot contain $\rtimes$ or $\ltimes$ as this would imply $L(G) = \emptyset$. Assume, then, that $S \subseteq \wp(\{0,1\})$. If $S$ contains both 0 and 1, it blocks every string and $L(G) = \emptyset$. If $S = \emptyset$, then $L(G) = \{0,1\}^*$. Neither is allowed because $G$ is non-trivial. The only possible values for $S$, then, are $\{0\}$ and $\{1\}$. In those cases, $G$ generates $L(\textit{every})$ and $L(\textit{no})$, respectively.

*Case 2* $T := \{1\}$. $S$ cannot be empty as $G$ must not generate $\{0,1\}^*$. Note that if $S$ contains $1^{k_0}, 1^{k_1}, 1^{k_2}, \ldots, 1^{k_n}$ such that $k_0 < k_1 < k_2 < k_n$, all $k_i$ with $i > 0$ can be removed without changing the language generated by $G$. Now suppose that $S$ consists only of $1^k$. Then $G$ generates the quantifier language of *at most k*. On the other hand, if $S$ contains $\rtimes 1^k \ltimes$, then $\rtimes 1^j \ltimes$ is also a member of $S$ for all $0 \leq j \leq k$ — otherwise,

$G$ would not be downward closed. Given such an $S$, $G$ generates the quantifier language of *at least m*, where $m$ is the largest $k$ such that $\rtimes 1^k \ltimes \in S$. If $S$ contains $n$-grams of both types, the generated language describes the quantifier *exactly n* or *between m and n* ($m \leq n$ holds because $G$ does not generate the empty language).

This fully exhausts the range of possible grammars. □

The class of monomorphemic quantifiers corresponds exactly to the list in the theorem above if one regards *many* and *few* as context-dependent instantiations of *between m and n*, includes *exactly n* as a basic meaning of numerals rather than a pragmatically derived one, and ignores existential import. If *between m and n* and *exactly n* are to be excluded from the list of monomorphemic quantifiers, an additional property is needed.

**Definition 7.** *A TSL grammar* $G := \langle S, T \rangle$ *over alphabet* $\{0,1\}$ *is* unidirectional *iff either* $S \subsetneq \{\rtimes\} \times \{0,1\}^* \times \{\ltimes\}$ *or* $S \subsetneq \{0,1\}^*$.

Unidirectional quantifiers only establish upper bounds (*at most n*) or lower bounds (*at least n*), but not both.

These additional restrictions on TSL grammars also prompt a strengthened version of Conjecture 1:

**Conjecture 2.** *If a natural language contains a monomorphemic quantifier Q, then* $L(Q)$ *is generated by a TSL grammar that is*

- *non-trivial, and*
- *truth affine, and*
- *domain agnostic, and*
- *downward closed and/or unidirectional.*

The mathematical aspects of Theorem 1 and Conjecture 2 are on solid ground. The contentious issue is what quantifiers one takes to be monomorphemic. Any one of the proposed classification schemes may ultimately turn out to be untenable, but even then the consequences would not be devastating. The conditions on TSL grammars would need to be altered, and there may be some monomorphemic quantifiers that exceed the expressivity of TSL. Nonetheless TSL is a very weak subclass of the regular languages that can easily accommodate a large number of natural language quantifiers. What more, most TSL grammars over binary strings define fairly simple and natural quantifiers, and a small number of intuitively pleasing restrictions narrow the range of TSL to at least a majority of monomorphemic quantifiers. All of this demonstrates that the subregular approach to generalized quantifiers provides

interesting new insights and a strong basis for future research, some facets of which I briefly outline in the next and final section.

## 5 Discussion and Open Ends

### 5.1 The Status of *Only*

Throughout the paper I implicitly adopted the received view that the set of monomorphemic quantifiers does not include *only*. In contrast to monomorphemic quantifiers, *only* is not conservative. Whereas $every(A,B)$ is true iff $every(A,A\cap B)$ is true, $only(A,B)$ may be false while $only(A,A\cap B)$ holds. This is illustrated by the following contrast:

(2) a. Every boy sleeps.
b. Every boy is a boy who sleeps.

(3) a. Only boys sleep.
b. Only boys are boys who sleep.

In addition, *only* is usually analyzed as some kind of adverbial marker rather than a determiner. So if one only considers generalized quantifiers that are determiners, *only* is excluded for the same reason as *half*.

Nevertheless *only* can be studied from the perspective of quantifier languages. The two examples below are logically equivalent, which shows that *only* has a meaning similar to *all*:

(4) a. Only professors attended the ceremony.
b. All ceremony attendants were professors.

The difference is that $all(A,B)$ holds iff $A\subseteq B$, whereas $only(A,B)$ is true iff $B\subseteq A$. So *all* and *only* only differ in the directionality of the subset relation. Suppose for the sake of argument that *all* is defined over binary strings of $A$ under $B$, whereas *only* is stated with respect to binary strings of $B$ under $A$. In other words, the binary strings for *all* and *only* are computed by $f_B^A$ and $f_A^B$, respectively. In this case, the different subset relations turn into exactly the same string condition: no string may contain any 0s. Consequently, $L(all)=L(only)$. From a computational perspective, the existence of *only* would then be entirely unsurprising since there is no reason why binary strings should have to be computed by $f_B^A$ instead of $f_A^B$.

However, this raises the question whether other quantifiers have comparable counterparts that operate with $f_A^B$ rather than $f_B^A$. To some extent that seems to be the case. A comparable reversal also takes place when *only* occurs with numerals, although multiple readings are available in these cases.

(5) a. Only five professors attended the ceremony.
b. Exactly five ceremony attendants were professors.
c. All ceremony attendants were professors, of which there were exactly five.

The paraphrase in (5b) shows once again the switch between $A$ and $B$. While *Exactly five professors attended the ceremony* corresponds to *exactly five*$(A,B)$, (5b) is equivalent to *exactly five*$(B,A)$. The reading in (5c) is more involved, but reversal is visible if one contrasts it with a more verbose version of (5a):

(6) a. Only professors attended the ceremony, and it was only five professors that attended the ceremony.
b. All ceremony attendants were professors, and exactly five ceremony attendants were professors.

Here once again the reversal of $A$ and $B$ is evident. An integral component of *only*, then, is to indicate a switch from $f_B^A$ to $f_A^B$.

Of course this still fails to address many essential issues. Why is *five* strengthened to *exactly five*? Why is there no language where $f_A^B$ is the basic function and usage of $f_B^A$ must be explictly indicated by a dedicated marker *ynlo*, the opposite of *only*? And why is *only*$(A,B)$ the analogue of *every*$(A,B)$ rather than, say, *some*$(A,B)$? The subregular perspective offers a tentative answer at least to the last question: it is reasonable to regard $L(\textit{every})$ as the simplest quantifier language, and it is intuitively appealing that *only* would be based on the simplest case. The reader might not find these speculations convincing, but this does not affect the main point: a very minor change in the definition, namely the switch from $f_B^A$ to $f_A^B$, greatly opens up the subregular approach without losing the core insights of this paper. A better computational understanding of $f_B^A$ and $f_A^B$ may be all that is needed for a unified treatment of *only* and the standard monomorphemic quantifiers.

### 5.2 Proportional *many*

The claim that all quantifiers that are monomorphemic and act as determiners have TSL quantifier languages rests on the assumption that no attested proportional quantifiers fall into this class of quantifiers. For *most*, the argument is that it is morphologically complex and thus not monomorphemic (Hackl 2009). Another problematic case is *half*, which is excluded on the basis that it is not a determiner. But this does not fully exhaust the set of potential counterexample because *many* also admits a proportional reading.[6]

6 I am indebted to Lucas Champollion for bringing this to my attention.

Consider the sentence below.

(7) Many actors are attractive.

Under the cardinal reading, ((7)) asserts that there is a large number of actors that are attractive. But one could also interpret ((7)) such that the ratio of attractive to unattractive actors is high. Since there is no reason to doubt that *many* is both a determiner and monomorphemic, the availability of a proportional reading conflicts with the conjecture that such quantifiers must have TSL quantifier languages.

But proportional *many* is not necessarily at odds with the TSL conjecture, as it depends on what one takes to be the cognitively central aspect of a quantifier. If one takes quantifiers to refer to different semantic objects, then the ambiguity of *many* is due to there being two quantifiers $Q$ and $Q'$ that just happen to both be pronounced *many*. However, if one takes the morphological word as the central object, then *many* is a single quantifier with multiple possible interpretations. Under the first view, the existence of proportional *many* contradicts the TSL conjecture. But the second view allows us to reinterpret the TSL conjecture as a condition on the make-up of monomorphemic quantifiers such that they must have at least one reading that is sufficiently simple to be TSL.

### 5.3 Learnability

One of the most important questions about generalized quantifiers is how their meaning can be acquired from linguistic input (see e.g. Paperno 2011). The semantic automata approach provides an immediate attack vector for this problem because the learnability of string languages is an intensely studied research area. With respect to the subregular hierarchy, it is known that even the class of strictly local languages at the very bottom of the hierarchy is not learnable in the limit from positive text. The proof is a simple corollary of Gold's (1967) theorem that no language class that contains all finite languages and at least one infinite language can be learned in this paradigm. It is not particularly hard to extend this proof to the restricted case of permutation-closed string languages. At face value, then, none of the quantifier languages discussed in this paper are learnable in the limit from positive text.

However, some learnability results do hold after all. First, we now know that *every* and *no* have strictly 1-local quantifier languages and, more importantly, these are the only permutation-closed, non-trivial strictly local string languages generated by domain agnostic grammars. This leaves only two strictly local quantifier languages: $1^*$ (*every*) and $0^*$ (*no*). This class is learnable by virtue of being finite, but beyond that it is also efficiently learnable since each language can be inferred from at most one string in the input:

| Language | Required Input Strings |
|---|---|
| $1^*$ | any member of $1^+$ |
| $0^*$ | any member of $0^+$ |

The fact that every finite class of languages is learnable also implies that the TSL quantifier languages are learnable as long as one puts an upper bound on the length of *n*-grams. However, this is too strong an assumption because a speaker who grasps the concept of *at least n* does so for every *n*. Unfortunately, the class of *at least n* quantifier languages, even considered in isolation, is not learnable in the limit from text. This time, non-learnability is a corollary of Angluin's (1980) subset theorem. Without further research, then, we only have a choice between positing an upper bound on the size of *n*-grams in order to obtain a trivial learnability result, or to shun such an upper bound at the prize of a rather uninformative non-learnability result.

That said, not all finite classes are the same with respect to learnability even though they are all learnable. Some finite classes are efficiently learnable because they still display some internal regularity, whereas others are too random for meaningful generalization from very little input. Every subclass of TSL that is limited to *n*-grams of a fixed size is an instance of so-called string-extension classes (Heinz 2010b, Kasprzik & Kötzing 2010, Heinz et al. 2012), and those are efficiently learnable. So the commitment to an upper bound for counting quantifiers may be unappealing, but it nonetheless reveals that natural language quantifiers contain a lot of internal regularity that can be exploited by learning algorithms.

Of course there are many other paradigms besides learning in the limit from positive text, e.g. PAC learning (Valiant 1984), which is explored for quantifiers in Paperno (2011). The advantage of studying quantifiers from a subregular perspective is that any learning paradigm that was developed for string languages can be extended to quantifiers, and that subregular classes are particularly easy to study this way thanks to their simplicity.

### 5.4 Connections to Existing Work

Quantification in natural languages is not limited to individuals, it also includes quantification over possible worlds, events, and points in time. The latter is of particular interest as it intersects with ongoing work by Fernando (2011, 2015, and references therein). Fernando represents time via strings of events and then associates temporal operators with specific string languages in a fashion that is very similar to how the semantic automata approach relates quantifiers to sets of binary strings. He then provides a finite-state calculus for composing the meaning of these operators. Unfortunately, the calculus relies on many closure properties of regular languages that do not hold for any of the weaker classes, e.g. closure under union,

complement, and concatenation. This makes it very hard to judge whether the power of regular languages is fully utilized. I conjecture that first-order logic is sufficient to describe the calculus, in which case Fernando's string languages would be at most star-free (the second highest class in the subregular hierarchy, see Fig. 1). The relevant conditions may even be in the intersection closure of TSL, a class that is also needed for selected phenomena in phonology according to De Santo (2017).

A very different issue is the psychological reality of the computational differences established in this paper. As discussed in Steinert-Threlkeld & Icard (2013), there is some evidence for complexity differences between quantifiers. If the automaton computing a quantifier language consumes a large amount of memory, this can be detected in neuro-linguistic experiments (McMillan et al. 2005). Whether one equates processing difficulty with memory usage or the place in the subregular hierarchy leads to different predictions. It would be interesting to see whether the predictions of the subregular perspective can be reconciled with the neuro-linguistic evidence, and whether different experimental paradigms such as self-paced reading or eye-tracking confirm the neuro-linguistic measurements.

## 6 Conclusion

This paper has extended the semantic automata perspective of generalized quantifiers beyond the familiar classes of the Chomsky hierarchy by considering the subregular complexity of quantifier languages. Many natural language quantifiers turned out to be very weak and occupy a place close to the bottom of the subregular hierarchy. In particular, *every* and *no* are simpler than *some* and *not all*, which in turn are simpler than numerals. All of these quantifiers belong to the class of tier-based strictly local languages, which also plays an important role in phonology, morphology, and syntax. This class excludes more complex quantifiers such as *an even number*, *most*, or *at least one third*.

The importance of tier-based strict locality is given additional support by its close correspondence to the set of monomorphemic quantifiers. With the exception of *most*, *half* and possibly *only*, all monomorphemic quantifiers can be defined by tier-based strictly local grammars. In addition, a few natural restrictions on these grammars limit them to exactly the class of attested monomorphemic quantifiers. Replacing the monolithic class of regular languages by a fine-grained hierarchy thus has made it possible to put an explicit and very tight upper bound on the power of monomorphemic quantifiers, decisively separating them from other natural language quantifiers. The relative ease with which these results were obtained suggests that other aspects of semantics may also be fruitfully studied from a subregular perspective.

## References


Aksënova, Alëna, Thomas Graf & Sedigheh Moradi. 2016. Morphotactics as tier-based strictly local dependencies. In *Proceedings of the 14th sigmorphon workshop on computational research in phonetics, phonology, and morphology*, 121–130. https://www.aclweb.org/anthology/W/W16/W16-2019.pdf.

Angluin, Dana. 1980. Inductive inference of formal languages from positive data. *Information and Control* 45. 117–135.

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

van Benthem, Johan. 1986. Semantic automata. In *Essays in logical semantics*, 151–176. Dordrecht: Springer. http://dx.doi.org/10.1007/978-94-009-4540-1_8.

Bird, Steven. 1995. *Computational phonology: A constraint-based approach*. Cambridge: Cambridge University Press.

Büchi, J. Richard. 1960. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 6. 66–92. http://dx.doi.org/10.1002/malq.19600060105.

Chandlee, Jane. 2016. Computational locality in morphological maps. Ms., Haverford College; under review at *Morphology*.

Chandlee, Jane & Jeffrey Heinz. 2016. Computational phonology. Ms., Haverford College and University of Delaware.

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

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

Culy, Christopher. 1985. The complexity of the vocabulary of Bambara. *Linguistics and Philosophy* 8. 345–351. http://dx.doi.org/10.1007/978-94-009-3401-6_14. https://doi.org/10.1007/978-94-009-3401-6_14.

De Santo, Aniello. 2017. Extending TSL languages: Conjunction as multiple tier-projection. Ms., Stony Brook University.

Fernando, Tim. 2011. Regular relations for temporal propositions. *Natural Language Engineering* 11. 163–184.

Fernando, Tim. 2015. The semantics of tense and aspect: a finite-state perspective. In Shalom Lappin & Chris Fox (eds.), *The handbook of contemporary semantic theory, second edition*, 203–236. Wiley.

Gold, E. Mark. 1967. Language identification in the limit. *Information and Control* 10. 447–474. http://dx.doi.org/10.1016/S0019-9958(67)91165-5. https://doi.org/10.1016/S0019-9958(67)91165-5.

Graf, Thomas. 2010. *Logics of phonological reasoning*: University of California, Los Angeles MA thesis. http://thomasgraf.net/doc/papers/

LogicsOfPhonologicalReasoning.pdf.
Graf, Thomas. 2017. The power of locality domains in phonology. *Phonology* 34. 1–21. http://dx.doi.org/10.1017/S0952675717000197. https://dx.doi.org/10.1017/S0952675717000197. In press.
Graf, Thomas & Jeffrey Heinz. 2015. Commonality in disparity: The computational view of syntax and phonology. Slides of a talk given at GLOW 2015, April 18, Paris, France.
Hackl, Martin. 2009. On the grammar and processing of proportional quantifiers: Most versus more than half. *Natural Language Semantics* 17(1). 63–98. http://dx.doi.org/10.1007/s11050-008-9039-x.
Heinz, Jeffrey. 2007. *Inductive learning of phonotactic patterns*: UCLA dissertation. http://www.linguistics.ucla.edu/general/dissertations/heinz-2007-UCLA-diss.pdf.
Heinz, Jeffrey. 2009. On the role of locality in learning stress patterns. *Phonology* 26. 303–351. http://dx.doi.org/10.1017/S0952675709990145. https://doi.org/10.1017/S0952675709990145.
Heinz, Jeffrey. 2010a. Learning long-distance phonotactics. *Linguistic Inquiry* 41. 623–661. http://dx.doi.org/10.1162/LING_a_00015. http://dx.doi.org/10.1162/LING_a_00015.
Heinz, Jeffrey. 2010b. String extension learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, 897–906. http://www.aclweb.org/anthology/P10-1092.pdf.
Heinz, Jeffrey. 2014. Culminativity times harmony equals unbounded stress. In Harry van der Hulst (ed.), *Word stress: Theoretical and typological issues*, 255–275. Cambridge, UK: Cambridge University Press.
Heinz, Jeffrey. 2015. The computational nature of phonological generalizations. Ms., University of Delaware. http://www.socsci.uci.edu/~lpearl/colareadinggroup/readings/Heinz2015BC_Typology.pdf.
Heinz, Jeffrey, Anna Kasprzik & Timo Kötzing. 2012. Learning in the limit with lattice-structured hypothesis spaces. *Theoretical Computer Science* 457. 111–127. http://dx.doi.org/10.1016/j.tcs.2012.07.017. https://doi.org/10.1016/j.tcs.2012.07.017.
Heinz, Jeffrey, Chetan Rawal & Herbert G. Tanner. 2011. Tier-based strictly local constraints in phonology. In *Proceedings of the 49th annual meeting of the association for computational linguistics*, 58–64. http://www.aclweb.org/anthology/P11-2011.
Huybregts, M. A. C. 1984. The weak adequacy of context-free phrase structure grammar. In Ger J. de Haan, Mieke Trommelen & Wim Zonneveld (eds.), *Van periferie naar kern*, 81–99. Dordrecht: Foris.

Joshi, Aravind. 1985. Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions? In David Dowty, Lauri Karttunen & Arnold Zwicky (eds.), *Natural language parsing*, 206–250. Cambridge: Cambridge University Press.

Kasprzik, Anna & Timo Kötzing. 2010. String extension learning using lattices. In Adrian-Horia Dediu, Henning Fernau & Carlos Martín-Vide (eds.), *Language and automata theory and applications: 4th international conference, LATA 2010, Trier, Germany, May 24-28, 2010*, 380–391. Berlin, Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-642-13089-2_32. http://dx.doi.org/10.1007/978-3-642-13089-2_32.

Keenan, Edward L. & Leonard M. Faltz. 1985. *Boolean semantics for natural language*. Dordrecht: Reidel.

Keenan, Edward L. & Denis Paperno. 2012. *Handbook of quantifiers in natural language*. Berlin: Springer.

Kobele, Gregory M. 2006. *Generating copies: An investigation into structural identity in language and grammar*: UCLA dissertation. http://home.uchicago.edu/~gkobele/files/Kobele06GeneratingCopies.pdf.

McMillan, Corey T., Robin Clark, Peachie Moore, Christian Devita & Murray Grossman. 2005. Neural basis for generalized quantifier comprehension. *Neuropsychologia* 43. 1729–1737.

McMullin, Kevin. 2016. *Tier-based locality in long-distance phonotactics: Learnability and typology*: Uniersity of British Columbia dissertation.

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

McNaughton, Robert & Seymour Papert. 1971. *Counter-free automata*. Cambridge, MA: MIT Press.

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

Michaelis, Jens & Marcus Kracht. 1997. Semilinearity as a syntactic invariant. In Christian Retoré (ed.), *Logical aspects of computational linguistics*, vol. 1328 Lecture Notes in Artifical Intelligence, 329–345. Springer. http://dx.doi.org/10.1007/BFb0052165. http://dx.doi.org/10.1007/BFb0052165.

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

Perrin, Dominique & Jean-Éric Pin. 2004. *Infinite words. Automata, semigroups, logic and games*. Amsterdam: Elsevier.

Peters, Stanley & Dag Westerståhl. 2006. *Quantifiers in language and logic*. Oxford University Press.

Pin, Jean-Eric. 1997. Syntactic semigroups. In *Handbook of language theory*, 679–764. Berlin: Springer.

Potts, Christopher & Geoffrey K. Pullum. 2002. Model theory and the content of OT constraints. *Phonology* 19(4). 361–393.

Pullum, Geoffrey K. & James Rogers. 2006. Animal pattern-learning experiments: Some mathematical background. Ms., Radcliffe Institute for Advanced Study, Harvard University.

Radzinski, Daniel. 1991. Chinese number names, tree adjoining languages, and mild context sensitivity. *Computational Linguistics* 17. 277–300. http://ucrel.lancs.ac.uk/acl/J/J91/J91-3002.pdf.

Rogers, James, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Vischer, David Wellcome & Sean Wibel. 2010. On languages piecewise testable in the strict sense. In Christan Ebert, Gerhard Jäger & Jens Michaelis (eds.), *The mathematics of language*, vol. 6149 Lecture Notes in Artificial Intelligence, 255–265. Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-642-14322-9_19. http://dx.doi.org/10.1007/978-3-642-14322-9_19.

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

Ruiz, José, Salvador España & Pedro García. 1998. Locally threshold testable languages in strict sense: Application to the inference problem. In Vasant Honavar & Giora Slutzki (eds.), *Grammatical inference: 4th international colloquium, ICGI-98 Ames, Iowa, USA, July 12–14, 1998*, 150–161. Berlin: Springer.

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

Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii & Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science* 88. 191–229. http://dx.doi.org/10.1016/0304-3975(91)90374-B. https://doi.org/10.1016/0304-3975(91)90374-B.

Shieber, Stuart M. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy* 8(3). 333–345. http://dx.doi.org/10.1007/BF00630917. http://dx.doi.org/10.1007/BF00630917.

Simon, Imre. 1975. Piecewise testable events. In H. Brakhage (ed.), *Automata theory and formal languages 2nd gi conference*, vol. 33 Lectures Notes in Computer Science, 214–222. Berlin: Springer. http://dx.doi.org/10.1007/3-540-07407-4_23. http://dx.doi.org/10.1007/3-540-07407-4_23.

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

Valiant, Leslie G. 1984. A theory of the learnable. *Communications of the ACM* 27. 1134–1142.

Thomas Graf
Stony Brook University
Department of Linguistics
Stony Brook, NY 11794-4376
mail@thomasgraf.net