Almost All Complex Quantifiers are Simple

Jakub Szymanik

Department of Philosophy, Utrecht University
Heidelberglaan 6, 3584 CS Utrecht, The Netherlands
j.szymanik@uva.nl

Abstract. We prove that PTIME generalized quantifiers are closed under Boolean operations, iteration, cumulation and resumption.

Key words: generalized quantifiers; computational complexity; polyadic quantifiers; Boolean combinations; iteration; cumulation; resumption

1 Introduction

Most research in generalized quantifier theory has been directed towards monadic quantification in natural language. The recent monograph [1] bears witness to this tendency, devoting more than 90% of its volume to the discussion of monadic quantifiers. It is then clear that definability and complexity of monadic quantifiers has been extensively studied. For example, it is known that monadic quantifiers definable in first-order logic, like "all" or "at least 7", are recognizable by acyclic finite-automata [2]; first-order logic enriched by all quantifiers of the form "divisible by n" (e.g., "even" and "odd") corresponds to the class of all regular languages [3]; and that proportional quantifiers, like "most", can be recognized by push-down automata. Those results suggest that the comprehension of monadic quantifiers in natural language should be relatively easy. Indeed, a corpus of empirical studies shows that the cognitive task of recognizing logical-value of sentences with monadic quantifiers is simple. In fact, recently the automata-theoretic model for processing monadic quantifiers has been confronted with the human comprehension and the results show that the model captures many important cognitive aspects of the problem (see e.g. [4, 5]). Those studies — linking computational complexity with cognitive processing — lead to natural interest in computational properties of polyadic quantification (multi-quantifier sentences).

In the case of polyadic quantifiers things look differently. In the logical literature one can find examples of natural language polyadic quantifiers with computationally intractable model-checking problems. Branching quantifiers are an important example. It has been observed that branching readings of natural language sentences define NP-complete problems (see [6–8]). For instance, consider the following sentences:

(1) Some book by every author is referred to in some essay by every critic.
(2) In my class most boys and most girls dated each other.

Sentences with branching interpretations are definitely not a typical part of everyday language (see e.g. [9]) but one can find less controversial examples of semantic constructions with high computational complexity. For example, some reciprocal sentences with quantified antecedents are NP-complete given a plausible interpretation (see [10]), e.g. the following sentence:

(3) Most parliament members refer to each other.

Additionally, collective quantification, like in sentence (4), can also lead to combinatorial explosion when the verification process is considered (see [11]).

(4) Most of the PhD students played Hold'em together.

Those results show that the direct understanding of some polyadic quantifiers in natural language might be difficult, if not impossible (see e.g. [12, 8]). However, one might ask whether those examples are representative for natural language?

In this paper we study the most common semantic constructions that turn simple quantifiers into complex ones: Boolean operations, iteration, cumulation and resumption. We prove that they do not increase computational complexity when applied to determiners. More precisely, PTIME quantifiers are closed under the application of those lifts. Most of the natural language determiners correspond to monadic quantifiers computable in polynomial time. This observation suggests that typically polyadic quantifiers in natural language are tractable and the intractable constructions mentioned in the previous paragraph and extensively studied in logic are rather exceptional.

1.1 Mathematical Preliminaries

Generalized Quantifiers Usually in linguistic semantics quantifiers are treated as relations between subsets of the universe. However, it is well-known, and often used in logic, that equivalently generalized quantifiers might be defined as classes of models (see e.g. [1]). The formal definition is as follows:

Definition 1. *Let $t = (n_1, \ldots, n_k)$ be a k-tuple of positive integers. A generalized quantifier of type t is a class Q of models of a vocabulary $\tau_t = \{R_1, \ldots, R_k\}$, such that R_i is n_i-ary for $1 \leq i \leq k$, and Q is closed under isomorphisms, i.e. if $\mathbb{M}$ and $\mathbb{M}'$ are isomorphic, then*

$$(\mathbb{M} \in \mathsf{Q} \iff \mathbb{M}' \in \mathsf{Q}).$$

If in the above definition for all i: $n_i \leq 1$, then we say that a quantifier is *monadic*, otherwise we call it *polyadic*.

Let us explain this definition further by giving a few examples. Consider a sentence of the form Every A *is* B, where A stands for poets and B for people having low self-esteem. The sentence is true if and only if $A \subseteq B$. Therefore, according to the definition, the quantifier "every" is of type (1, 1) and corresponds to the class of models (M, A, B) in which $A \subseteq B$. For the same reasons the quantifier "an even number of" corresponds to the class of models in which

the cardinality of $A \cap B$ is an even number. Finally, let us consider the quantifier "most" of type (1, 1). The sentence Most *As are* B is true if and only if $card(A \cap B) > card(A - B)$ and therefore the quantifier corresponds to the class of models where this inequality holds.

Therefore, formally speaking:

$$\begin{aligned}
\forall &= \{(M, A) \mid A = M\}.\\
\exists &= \{(M, A) \mid A \subseteq M \text{ and } A \neq \emptyset\}.\\
\mathsf{Every} &= \{(M, A, B) \mid A, B \subseteq M \text{ and } A \subseteq B\}.\\
\mathsf{Even} &= \{(M, A, B) \mid A, B \subseteq M \text{ and } card(A \cap B) \text{ is even}\}.\\
\mathsf{Most} &= \{(M, A, B) \mid A, B \subseteq M \text{ and } card(A \cap B) > card(A - B)\}.
\end{aligned}$$

Quantifiers in Finite Models In the paper we are interested in finite models as arguably it is enough to model typical meanings of natural language expressions (see e.g. [5] for a discussion). Finite models can be encoded as finite strings over some vocabulary as follows. Let $\mathcal{K}$ be a class of finite models over some fixed vocabulary τ. We want to treat $\mathcal{K}$ as a problem (language) over the vocabulary τ. To do this we need to code τ-models as finite strings. We can assume that the universe of a model $\mathbb{M} \in \mathcal{K}$ consists of natural numbers: $U = \{1, \ldots, n\}$. A natural way of encoding a model $\mathbb{M}$ (up to isomorphism) is by listing its universe, U, and storing the interpretation of the symbols in τ by writing down their truth-values on all tuples of objects from U.

Definition 2. *Let $\tau = \{R_1, \ldots, R_k\}$ be a relational vocabulary and $\mathbb{M}$ a τ-model of the following form: $\mathbb{M} = (U, R_1^M, \ldots, R_k^M)$, where $U = \{1, \ldots, n\}$ is the universe of model $\mathbb{M}$ and $R_i^M \subseteq U^{n_i}$ is an n_i-ary relation over U, for $1 \leq i \leq k$. We define a binary encoding for τ-models. The code for $\mathbb{M}$ is a word over $\{0, 1, \#\}$ of length $O((card(U))^c)$, where c is the maximal arity of the predicates in τ (or $c = 1$ if there are no predicates).*

The code has the following form:

$$\tilde{n}\#\tilde{R_1^M}\#\ldots\#\tilde{R_n^M}, \text{ where:}$$

- *$\tilde{n}$ is the part coding the universe of the model and consists of n 1s.*
- *$\tilde{R_i^M}$ — the code for the n_i-ary relation R_i^M — is an n^{n_i}-bit string whose j-th bit is 1 iff the j-th tuple in U^{n_i} (ordered lexicographically) is in R_i^M.*
- *$\#$ is a separating symbol.*

Therefore, according to Definition 1, generalized quantifiers can be treated as classes of such finite strings, i.e., languages. Now we can easily fit the notions into the descriptive complexity paradigm (see e.g. [13]).

Definition 3. *By the complexity of a quantifier* Q *we mean the computational complexity of the corresponding class of finite models.*

For example, consider a quantifier of type (1, 2): a class of finite colored graphs of the form $\mathbb{M} = (M, A^M, R^M)$. Let us take a model of this form, $\mathbb{M}$, and a quantifier Q. Our computational problem is to decide whether $\mathbb{M} \in \mathsf{Q}$; or equivalently, to solve the query whether $\mathbb{M} \models \mathsf{Q}[A, R]$. This can simply be viewed as the model-checking problem for quantifiers.

In the following chapter we investigate the computational complexity of model checking problem for polyadic quantifiers commonly occuring in natural language. But before we turn to it we need to recall some basic concepts of the computational complexity theory (see e.g. [14] for an extensive treatment).

Complexity Classes Let $f : \omega \longrightarrow \omega$ be a natural number function. TIME(f) is the class of languages (problems) which can be recognized by a deterministic Turing machine in time bounded by f with respect to the length of the input. In other words, $L \in$ TIME(f) if there exists a deterministic Turing machine such that for every $x \in L$, the computation path of M on x is shorter than $f(n)$, where n is the length of x. TIME(f) is called a *deterministic computational complexity class*. A *non-deterministic complexity class*, NTIME(f), is the class of languages L for which there exists a non-deterministic Turing machine M such that for every $x \in L$ all branches in the computation tree of M on x are bounded by $f(n)$ and moreover M decides L. One way of thinking about a non-deterministic Turing machine bounded by f is that it first guesses the right answer and then deterministically in a time bounded by f checks if the guess is correct.

SPACE(f) is the class of languages which can be recognized by a deterministic machine using at most $f(n)$ cells of the working-tape. NSPACE(f) is defined analogously.

Below we define some well-known complexity classes, i.e., the sets of languages of related complexity. In other words, we can say that a complexity class is the set of problems that can be solved by a Turing machine using $O(f(n))$ of time or space resource, where n is the size of the input.

Definition 4.
- LOGSPACE $= \bigcup_{k\in\omega}$ SPACE($k \log n$)
- PTIME $= \bigcup_{k\in\omega}$ TIME(n^k)
- NP $= \bigcup_{k\in\omega}$ NTIME(n^k)

If $L \in$ NP, then we say that *L is decidable (computable, solvable) in non-deterministic polynomial time* and likewise for other complexity classes.

Moreover, we will need a concept of relativization defined via oracle machines. An oracle machine can be described as a Turing machine with a black box, called an oracle, which is able to decide certain decision problems in a single step. More precisely, an oracle machine has a separate write-only oracle tape for writing down queries for the oracle. In a single step, the oracle computes the query, erases its input, and writes its output to the tape.

Definition 5. *If $\mathcal{B}$ and $\mathcal{C}$ are complexity classes, then $\mathcal{B}$ relativized to $\mathcal{C}$, $\mathcal{B}^C$, is the class of languages recognized by oracle machines which obey the bounds defining $\mathcal{B}$ and use an oracle for problems belonging to $\mathcal{C}$.*

The question whether PTIME is strictly contained in NP is the famous Millennium Problem — one of the most fundamental problems in theoretical computer science, and in mathematics in general. The importance of this problem reaches well outside the theoretical sciences as the problems in NP are usually taken to be *intractable* or *not efficiently computable* as opposed to the problems in P which are conceived of as *efficiently solvable*. In the paper we take this distinction for granted and investigate semantic constructions in natural language from that perspective.

2 Complexity of Polyadic GQs in Language

One way to deal with polyadic quantification in natural language is to define it in terms of monadic quantifiers using Boolean combinations and so-called *polyadic lifts*. Below we recall definitions of the Boolean combinations of quantifiers and some well-known polyadic lifts: iteration, cumulation, and resumption (see e.g. [15]). Next we observe that they do not increase the computational complexity of quantifiers.

2.1 Boolean Combinations

To account for complex noun phrases, like those occurring in sentences (5)–(8), we define disjunction, conjunction, outer negation (complement) and inner negation (post-complement) of generalized quantifiers.

(5) At least 5 or at most 10 departments can win EU grants. (disjunction)
(6) Between 100 and 200 students started in the marathon. (conjunction)
(7) Not all students passed. (outer negation)
(8) All students did not pass. (inner negation)

Definition 6. *Let* Q, Q' *be generalized quantifiers, both of type* $(n_1, \ldots, n_k)$. *We define:*

$$(\mathsf{Q} \wedge \mathsf{Q}')_M[R_1, \ldots, R_k] \iff \mathsf{Q}_M[R_1, \ldots, R_k] \textit{ and } \mathsf{Q}'_M[R_1, \ldots, R_k] \text{ (conjunction)}$$

$$(\mathsf{Q} \vee \mathsf{Q}')_M[R_1, \ldots, R_k] \iff \mathsf{Q}_M[R_1, \ldots, R_k] \textit{ or } \mathsf{Q}'_M[R_1, \ldots, R_k] \text{ (disjunction).}$$

$$(\neg\mathsf{Q})_M[R_1, \ldots, R_k] \iff \textit{not } \mathsf{Q}_M[R_1, \ldots, R_k] \text{ (complement)}$$

$$(\mathsf{Q}\neg)_M[R_1, \ldots, R_k] \iff Q_M[R_1, \ldots, R_{k-1}, M - R_k] \text{ (post-complement)}$$

2.2 Iteration

The Fregean nesting of first-order quantifiers, e.g., $\forall\exists$, can be applied to any generalized quantifier by means of iteration. For example, iteration may be used to express the meaning of the following sentence in terms of its constituents.

(9) Most logicians criticized some papers.

The above sentence is true (under one interpretation) iff there is a set containing most logicians such that every logician from that set criticized at least one paper, or equivalently:

$$\mathsf{It}(\mathsf{Most}, \mathsf{Some})[\text{Logicians, Papers, Criticized}].$$

Of course the sentence can have a different reading corresponding to other lifts than iteration. We will introduce another possibility in Section 2.3. But first let us define iteration precisely.

Definition 7. *Let* Q *and* Q' *be generalized quantifiers of type (1, 1). Let* A, B *be subsets of the universe and* R *a binary relation over the universe. Suppressing the universe, we will define the iteration operator as follows:*

$$\mathsf{It}(\mathsf{Q}, \mathsf{Q}')[A, B, R] \iff \mathsf{Q}[A, \{a \mid \mathsf{Q}'[B, R_{(a)}]\}],$$

where $R_{(a)} = \{b \mid R(a, b)\}$.

Therefore, the iteration operator produces polyadic quantifiers of type (1, 1, 2) from two monadic quantifiers of type (1, 1). The definition can be extended to cover iteration of monadic quantifiers with an arbitrary number of arguments (see e.g. [1], p. 347).

2.3 Cumulation

Consider the following sentence:

(10) Eighty professors taught sixty courses at ESSLLI'08.

The analysis of this sentence by iteration of the quantifiers "eighty" and "sixty" implies that there were $80 \times 60 = 4800$ courses at ESSLLI. Therefore, obviously this is not the meaning we would like to account for. This sentence presumably means neither that each professor taught 60 courses ($\mathsf{It}(80, 60)$) nor that each course was taught by 80 professors ($\mathsf{It}(60, 80)$). In fact, this sentence is an example of so-called cumulative quantification, saying that each of the 80 professors taught at least one from 60 course and each of the courses was taught by at least one professor. Cumulation is easily definable in terms of iteration and the existential quantifier as follows.

Definition 8. *Let* Q *and* Q' *be generalized quantifiers of type (1, 1).* A, B *are subsets of the universe and* R *is a binary relation over the universe. Suppressing the universe we will define the cumulation operator as follows:*

$$\mathsf{Cum}(\mathsf{Q}, \mathsf{Q}')[A, B, R] \iff \mathsf{It}(\mathsf{Q}, \mathsf{Some})[A, B, R] \wedge \mathsf{It}(\mathsf{Q}', \mathsf{Some})[B, A, R^{-1}].$$

2.4 Resumption

The next lift we are about to introduce — resumption (vectorization) — has found many applications in theoretical computer science (see e.g. [13]). The idea here is to lift a monadic quantifier in such a way as to allow quantification over tuples. This is linguistically motivated when ordinary natural language quantifiers are applied to pairs of objects rather than individuals, for instance:

(11) Most twins never separate.

Moreover, resumption is useful for interpretation of certain cases of adverbial quantification (see e.g. [1], Ch. 10.2).

Below we give a formal definition of the *resumption* operator.

Definition 9. *Let* Q *be any monadic quantifier with* n *arguments,* U *a universe, and* $R_1, \ldots, R_n \subseteq U^k$ *for* $k \geq 1$*. We define the resumption operator as follows:*

$$\mathsf{Res}^k(\mathsf{Q})_U[R_1, \ldots, R_n] \iff (\mathsf{Q})_{U^k}[R_1, \ldots, R_n].$$

That is, $\mathsf{Res}^k(\mathsf{Q})$ is just Q applied to a universe, U^k, containing k-tuples. In particular, $\mathsf{Res}^1(\mathsf{Q}) = \mathsf{Q}$. Clearly, one can use $\mathsf{Res}^2(\mathsf{Most})$ to express the meaning of sentence (11).

2.5 PTIME GQs are Closed under It, Cum, and Res

When studying the computational complexity of quantifiers a natural question arises in the context of polyadic lifts. Do they increase complexity? For example, is it possible that two tractable determiners can be turned into an intractable quantifier?

We show that PTIME computable quantifiers are closed under Boolean combinations and the three lifts defined above. As we are interested in the strategies people may use to comprehend quantifiers we show a direct construction of the relevant procedures. In other words, we show how to construct a polynomial model-checker for our polyadic quantifiers from PTIME Turing machines computing monadic determiners.

Proposition 1 *Let* Q *and* Q' *be monadic quantifiers computable in polynomial time with respect to the size of a universe. Then the quantifiers:* (1) $\neg\mathsf{Q}$*;* (2) $\mathsf{Q}\neg$*;* (3) $\mathsf{Q} \wedge \mathsf{Q}'$*;* (4) $\mathsf{It}(\mathsf{Q}, \mathsf{Q}')$*;* (5) $\mathsf{Cum}(\mathsf{Q}, \mathsf{Q}')$*;* (6) $\mathsf{Res}(\mathsf{Q})$ *are PTIME computable.*

Proof. Let us assume that there are Turing machines M and M' computing quantifiers Q and Q', respectively. Moreover M and M' work in polynomial time with respect to any finite universe U.

(1) A Turing machine computing $\neg\mathsf{Q}$ is like M. The only difference is that we change accepting states into rejecting states and *vice versa*. In other words, we accept $\neg\mathsf{Q}$ whenever M rejects Q and reject whenever M accepts. The working time of so-defined new Turing machine is exactly the same as the working time of machine M. Hence, the outer negation of PTIME quantifiers can be recognized in polynomial time.

(2) Recall that on a given universe U we have the following equivalence: $(\mathsf{Q}\neg)_U[R_1,\ldots,R_k] \iff \mathsf{Q}_U[R_1,\ldots,R_{k-1},U-R_k]$. Therefore, for the inner negation of a quantifier it suffices to compute $U-R_k$ and then use the polynomial Turing machine M on the input $\mathsf{Q}_U[R_1,\ldots,R_{k-1},U-R_k]$.

(3) To compute $\mathsf{Q}\wedge\mathsf{Q}'$ we have to first compute Q using M and then Q' using M'. If both machines halt in an accepting state then we accept. Otherwise, we reject. This procedure is polynomial, because the sum of the polynomial bounds on working time of M and M' is also polynomial.

(4) Recall that $\mathsf{It}(\mathsf{Q},\mathsf{Q}')[A,B,R] \iff \mathsf{Q}[A,A']$, where $A'=\{a|\mathsf{Q}'[B,R_{(a)}]\}$, for $R_{(a)}=\{b|R(a,b)\}$. Notice that for every a from the universe, $R_{(a)}$ is a monadic predicate. Now, to construct A' in polynomial time we execute the following procedure for every element from the universe. We initialize $A'=\emptyset$. Then we repeat for each a from the universe the following: Firstly we compute $R_{(a)}$. Then using the polynomial machine M' we compute $\mathsf{Q}'[B,R_{(a)}]$. If the machine accepts, then we add a to A'. Having constructed A' in polynomial time we just use the polynomial machine M to compute $\mathsf{Q}[A,A']$.

(5) Notice that cumulation is defined in terms of iteration and existential quantifier (see Definition 8). Therefore, this point follows from the previous one.

(6) To compute $\mathsf{Res}^k(\mathsf{Q})$ over the model $\mathbb{M}=\{\{1,\ldots,n\},R_1,\ldots,R_n\}$ for a fixed k, we just use the machine M with the following input $\tilde{n^k}\#\tilde{R_1}\#\ldots\#\tilde{R_n}$ instead of $\tilde{n}\#\ldots$ Recall Definition 2.

Additionally, let us give an argument that the above proposition holds for all generalized quantifiers not only for the monadic ones. Notice that the Boolean operations as well as iteration and cumulation are definable in first-order logic. Recall that the model-checking problem for first-order sentences is in LOGSPACE $\subseteq$ PTIME (see e.g. [13]). Let A be a set of generalized quantifiers of any type from a given complexity class $\mathcal{C}$. Then the complexity of model-checking for sentences from FO(A) is in LOGSPACE$^{\mathcal{C}}$ (deterministic logarithmic space with an oracle from $\mathcal{C}$). One simply uses a LOGSPACE Turing machine to decide the first-order sentences, evoking the oracle when a quantifier from A appears. Therefore, the complexity of Boolean combinations, iteration and cumulation of PTIME generalized quantifiers has to be in LOGSPACE$^{\text{PTIME}}$ = PTIME.

The case of the resumption operation is slightly more complicated. Resumption is not definable in first-order logic for all generalized quantifiers (see [16]). However, notice that our argument given in point (6) of the proof do not make use of any assumption about the arity of R_i. Therefore, the same proof works for resumption of polyadic quantifiers. The above considerations allow us to formulate the following theorem which is the generalization of the previous proposition.

Theorem 1 *Let* Q *and* Q' *be generalized quantifiers computable in polynomial time with respect to the size of a universe. Then the quantifiers:* (1) $\neg\mathsf{Q}$*;* (2) $\mathsf{Q}\neg$*;* (3) $\mathsf{Q}\wedge\mathsf{Q}'$*;* (4) $\mathsf{It}(\mathsf{Q},\mathsf{Q}')$*;* (5) $\mathsf{Cum}(\mathsf{Q},\mathsf{Q}')$*;* (6) $\mathsf{Res}(\mathsf{Q})$ *are PTIME computable.*

3 Conclusion

We have shown that PTIME quantifiers are closed under Boolean operations as well as under the polyadic lifts occurring frequently in natural language. In other words, these operations do not increase the computational complexity of quantifiers. As we can safely assume that most of the simple determiners in natural language are PTIME computable the semantics of the polyadic quantifiers studied above is tractable. This seems to be good news for the computational theory of natural language processing.

References

1. Peters, S., Westerståhl, D.: Quantifiers in Language and Logic. Clarendon Press, Oxford (2006)
2. van Benthem, J.: Essays in logical semantics. Reidel (1986)
3. Mostowski, M.: Computational semantics for monadic quantifiers. Journal of Applied Non-Classical Logics **8** (1998) 107–121
4. McMillan, C.T., Clark, R., Moore, P., Devita, C., Grossman, M.: Neural basis for generalized quantifier comprehension. Neuropsychologia **43** (2005) 1729–1737
5. Szymanik, J., Zajenkowski, M.: Comprehension of simple quantifiers. Empirical evaluation of a computational model. Cognitive Science **in press** (2009)
6. Mostowski, M., Wojtyniak, D.: Computational complexity of the semantics of some natural language constructions. Annals of Pure and Applied Logic **127**(1-3) (2004) 219–227
7. Sevenster, M.: Branches of imperfect information: logic, games, and computation. PhD thesis, Universiteit van Amsterdam (2006)
8. Szymanik, J.: Quantifiers in TIME and SPACE. Computational Complexity of Generalized Quantifiers in Natural Language. PhD thesis, Universiteit van Amsterdam (2009)
9. Gierasimczuk, N., Szymanik, J.: Branching quantification vs. two-way quantification. Journal of Semantics **26**(4) (November 2009) 367–392
10. Szymanik, J.: The computational complexity of quantified reciprocals. In Bosch, P., Gabelaia, D., Lang, J., eds.: Lecture Notes in Artificial Inteligence. Volume 5422., Springer (2008) 139–152
11. Kontinen, J., Szymanik, J.: A remark on collective quantification. Journal of Logic, Language and Information **17**(2) (April 2008) 131–140
12. Frixione, M.: Tractable competence. Minds and Machines **11**(3) (2001) 379–397
13. Immerman, N.: Descriptive Complexity. Texts in Computer Science. Springer (November 1998)
14. Papadimitriou, C.H.: Computational Complexity. Addison Wesley (November 1993)
15. van Benthem, J.: Polyadic quantifiers. Linguistics and Philosophy **12**(4) (1989) 437–464
16. Hella, L., Väänänen, J., Westerståhl, D.: Definability of polyadic lifts of generalized quantifiers. Journal of Logic, Language and Information **6**(3) (July 1997) 305–335