Constructing Semantic Automata for Quantifier Iterations

Sarah McWhirter
Center for Biostatistics
in AIDS Research
Harvard T.H. Chan School of Public Health
651 Huntington Avenue, FXB 543
Boston, MA 02115
sarahjmcwhirter@gmail.com

Jakub Szymanik
Institute for Logic, Language, and Computation
University of Amsterdam
Science Park 107
1098 XG Amsterdam
The Netherlands
j.k.szymanik@uva.nl

Abstract

We expand on recent work extending the semantic automata model of quantifier verification to iteration. We demonstrate a simple and intuitive method to construct a minimal iteration DFA from two DFA recognizing monadic regular quantifier languages and prove that deterministic CFL are also closed under quantifier iteration. We also touch upon the relation between computational results and linguistic discussion of Frege boundary.

1 Introduction

Nearly thirty years ago, van Benthem first introduced the notion of semantic automata, uniting generalized quantifier theory (GQT) and formal language theory in an elegant and powerful way. The basic idea is to use insights of GQT to identify a natural language quantifier with an automaton that, in a precise sense, recognizes the models in which it is true, letting us consider the quantifier as a procedure for checking whether it holds. This marriage enables the use of the Chomsky hierarchy as a measure of complexity of quantifiers, which "turns out to make eminent sense, both in its coarse and fine structure" [2].

Semantic automata have not only led to many observations interesting in their own theoretical right, but also to myriad insights in cognitive modeling and formal learning based on the idea of meaning as algorithm [9, 5, 25, 4]. A steady flow of research was inspired by semantic automata in the following decades, but very little work has been done toward broadening the model to address more than simple monadic quantifiers, though the definability of polyadic quantification is much-studied. Szymanik [24] provides a computational complexity (Turing machine-based) perspective on polyadic quantifiers, showing that some of those natural language constructions are polynomial-time closed and others are NP-hard. Those results prompt the question: can we also obtain automata characterizations for some polyadic quantifiers?

This paper is inspired by a recent (2013) answer to that question by Steinert-Threlkeld and Icard III [23], exploring semantic automata for iterated quantifiers. They show regular and context-free quantifier languages are closed under iteration; however, their proposed computational model is unnecessarily powerful. We present a construction that is appropriately powerful. Also in 2013, Kanazawa [11] answered an open question in [19] characterizing the class of quantifiers recognized by deterministic pushdown automata by their corresponding semilinear sets. As it turns out, it is rather difficult to form simple natural language quantifiers that go beyond this characterization. Given this new result, it is interesting to ask whether this natural subset of context-free languages is closed under quantifier iteration. We prove that it is indeed the case.

2 Prerequisites

2.1 Generalized Quantifiers

In this section we recall a few basic concepts of GQT and refer readers to [20] for more information. GQT applied to natural language treats determiners as relations between the denotations of other constituents of a sentence. For example, every is the inclusion relation: *Every student wrote a thesis* is true just in case every individual in the set of students is also in the set of thesis-writers. We can write the meanings of the quantifiers as:

$$\begin{array}{lcl} \mathsf{every} & = & \{(M, A, B) : A \subseteq B\} \\ \mathsf{at\ least\ three} & = & \{(M, A, B) : \mid A \cap B \mid \geq 3\} \end{array}$$

These are simple examples, but a quantifier may denote a relation between any number of relations of any arity. Mostowski [18] first introduced the general notion of a unary quantifier, binding a single variable in a formula similarly to $\forall$ and $\exists$ in standard logic. Lindström extended this to arbitrary types.

Definition 2.1. [16] A Lindström quantifier Q of type $\langle n_1, \ldots, n_k \rangle$ is a class of models $\mathcal{M} = (M, R_1, \ldots, R_k)$ with the R_i n_i-ary that is closed under isomorphism.

Monadic quantifiers (i.e. of type $\langle 1, \ldots, 1 \rangle$) are sufficient to analyze simple sentences following the schema *Q_1 A are B*, as in *Every Olympian is an athlete*. However, natural language is full of examples of polyadic quantification, such as the following *iteration*:

Half the students passed every class.

Taking S as the set of students, C as the set of classes, and $P = \{(s, c) : s \text{ passed } c\}$, we can give the truth conditions of this sentence as:

$$\begin{array}{c} (\mathsf{half} \cdot \mathsf{every})(S, C, P) \\ \Leftrightarrow \mathsf{half}(S, \{s : \mathsf{every}(C, P_s)\}) \end{array}$$

where P_s is the set $\{c : (s, c) \in P\}$. The iteration of half and every yields a new quantifier $\mathsf{half} \cdot \mathsf{every}$ which takes two sets and a binary relation between them as arguments.

Definition 2.2. Let Q_1 and Q_2 both be of type $\langle 1, 1 \rangle$. $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ is the type $\langle 1, 1, 2 \rangle$ quantifier such that for all $A, B \subseteq M$ and $R \subseteq M^2$:

$$(\mathsf{Q}_1 \cdot \mathsf{Q}_2)(A, B, R) \Leftrightarrow \mathsf{Q}_1(A, \{a : \mathsf{Q}_2(B, R_a)\})$$

The iteration of three type $\langle 1, 1 \rangle$ quantifiers creates a type $\langle 1, 1, 1, 3 \rangle$ quantifier, and so forth.

Natural language determiners are generally taken to satisfy certain semantic universals [1] yielding so-called CE-quantifiers:

- Q of type $\langle 1, 1 \rangle$ satisfies extensionality (EXT) if and only if for all $A, B \subseteq M$ and $M \subseteq M'$:
$$\mathsf{Q}_M(A, B) \Leftrightarrow \mathsf{Q}_{M'}(A, B)$$
- Q of type $\langle 1, 1 \rangle$ is conservative (CONS) if and only if for all M and $A, B \subseteq M$:
$$\mathsf{Q}_M(A, B) \Leftrightarrow \mathsf{Q}_M(A, A \cap B)$$
- Q of type $\langle 1, 1 \rangle$ satisfies isomorphism closure (ISOM)[1] if and only if for all $A, B \subseteq M$ and $A', B' \subseteq M'$, if $(M, A, B) \cong (M', A', B')$:
$$\mathsf{Q}_M(A, B) \Leftrightarrow \mathsf{Q}_{M'}(A', B')$$

Definition 2.3. For Q a type $\langle 1, 1 \rangle$ CE quantifier, define Q^c by:

$$\begin{array}{c} \mathsf{Q}^c(x, y) \Leftrightarrow \exists M \text{ and } A, B \subseteq M \text{ s.t.} \\ \mid A - B \mid = x, \mid A \cap B \mid = y \text{ and } \mathsf{Q}_M(A, B) \end{array}$$

Theorem 2.4 ([2])**.** Let Q of type $\langle 1, 1 \rangle$ be a CE-quantifier. Then for all M and $A, B \subseteq M$:

$$\mathsf{Q}(A, B) \Leftrightarrow \mathsf{Q}^c(\mid A - B \mid, \mid A \cap B \mid)$$

2.2 Regular and Deterministic Context-free Languages

We assume familiarity with formal language theory [10], but recall a few definitions and key results for later reference, mostly following [21].

Definition 2.5. A deterministic finite automaton (DFA) A is a five-tuple $(\mathcal{Q}, \Sigma, \delta, s, F)$ where $\mathcal{Q}$ is a finite set of states, Σ is an input alphabet, δ is a function from $\mathcal{Q} \times \Sigma$ to $\mathcal{Q}$, $s \in \mathcal{Q}$ is the start state, and $F \subseteq \mathcal{Q}$ is a set of final states.

A DFA is often graphically represented as a set of nodes (the states of the machine, with the start

[1]Note that isomorphism closure is part of our definition of generalized quantifier from the outset.

state indicated by an ingoing arrow with no source, and the final states doubly circled) with labeled, directed edges between them (representing the transition function). An edge from q to p labeled a means that $\delta(q,a) = p$. We can extend δ to be defined for entire strings in the obvious way, setting $\delta(q,w) = \delta(\delta(q,a),v)$ where $w = av$ and a is a single symbol of Σ. The language of A is the set of strings w such that a *run* of A (a computation beginning in s, reading w and transitioning according to δ) ends in a final state:

$$L(A) = \{w : \delta(s,w) \in F\}$$

The set of languages accepted by some DFA are the *regular languages* (REG). REG has nice closure properties, including closure under concatenation, substitution, and complementation. These results are all easily proven via automata construction; however, while the first two generally result in *non-deterministic* finite automata due to the addition of ϵ transitions, the complement is obtained by switching final and non-final states.

Deterministic context-free languages (DCFLs) are a proper subclass of context-free languages.

Definition 2.6. A deterministic pushdown automaton (DPDA) M is given by a six-tuple $(\mathcal{Q},\Sigma,\Gamma,Z_0,\delta,s,F)$ where:

- δ is a function from $\mathcal{Q} \times \Sigma \times \Gamma$ to $(\mathcal{Q} \times \Gamma) \cup \{\varnothing\}$ such that the following condition holds for every $q \in \mathcal{Q}$, $a \in \Sigma$, and $x \in \Gamma$:

  exactly one of $\delta(q,a,x), \delta(q,a,\epsilon), \delta(q,\epsilon,x)$ and $\delta(q,\epsilon,\epsilon)$ is non-empty.

This ensures that M always has exactly one move per configuration (is deterministic).

DPDA extend DFA with a stack (the last-in-first-out data structure) having push and pop operations. They may accept by final state or empty stack, but these notions are equivalent for end-marked languages.

Like REG, CFL is closed under concatenation and substitution, but not complementation. DCFL is closed under complementation (see Lemma 5.1), but not substitution.

2.3 Semantic Automata for Monadic Quantifiers

Recall that if a quantifier Q satisfies CONS, EXT, and ISOM, then it has an equivalent representation as a binary relation on natural numbers Q^c such that

$$\mathsf{Q}(A,B) \Leftrightarrow \mathsf{Q}^c(\mid A - B \mid, \mid A \cap B \mid)$$

Since the truth of $\mathsf{Q}(A,B)$ depends only on the cardinalities of A and $A \cap B$, we can record all the information relevant to its evaluation as a string of 0's and 1's with one symbol per element a in A: if a is in $A \cap B$, record a 1, otherwise record 0 (a is in $A - B$). Formally, we can define the following translation function.

Definition 2.7. Let $\mathcal{M} = \langle M, A, B\rangle$ be a model, $\vec{a}$ an enumeration of A, and $n = \mid A \mid$. We define $\tau(\vec{a},B) \in \{0,1\}^n$ by

$$(\tau(\vec{a},B))_i = \begin{cases} 0 & a_i \in A - B \\ 1 & a_i \in A \cap B \end{cases}$$

For a string $s \in \{0,1\}^*$ generated by $\tau(\vec{a},B)$, let $\#_0(s)$ denote the number of 0's in s and $\#_1(s)$ the number of 1's. Then we have

$$\begin{gathered} (\#_0(s), \#_1(s)) \in \mathsf{Q}^c \\ \Leftrightarrow (\mid A - B \mid, \mid A \cap B \mid) \in \mathsf{Q}^c \\ \Leftrightarrow \mathsf{Q}(A,B) \end{gathered}$$

Since we have a correspondence between models and strings, we can take the set of strings corresponding to the set of models where Q is true to constitute the *language* of Q:

$$\mathcal{L}_{\mathsf{Q}} = \{s \in \{0,1\}^* : (\#_0(s), \#_1(s)) \in \mathsf{Q}^c\}$$

For example:

$$\begin{array}{lcl} \mathcal{L}_{\text{every}} & = & \{s \in \{0,1\}^* : \#_0(s) = 0\} \\ \mathcal{L}_{\text{exactly three}} & = & \{s \in \{0,1\}^* : \#_1(s) = 3\} \\ \mathcal{L}_{\text{even}} & = & \{s \in \{0,1\}^* : \#_1(s) \bmod 2 = 0\} \\ \mathcal{L}_{\text{half}} & = & \{s \in \{0,1\}^* : \#_1(s) = \#_0(s)\} \end{array}$$

This opens up the application of automata theory to generalized quantification.

Theorem 2.8. [2] The first-order (FO) definable quantifiers are precisely those which can be recognized by permutation-invariant acyclic finite state machines.

Theorem 2.9. [19] Finite automata accept the class of all monadic quantifiers definable in FO logic extended with all divisibility quantifiers.

For example, exactly three and every, which are FO definable, are accepted by the respective automata in Figure 2. To account for the "counting modulo" quantifiers such as every, which are not FO definable, we need automata with loops. We will refer to quantifiers with languages accepted by finite automata as *regular quantifiers*.

Theorem 2.10. [2] Every PDA-computable quantifier is FO additively definable.

Theorem 2.11. [2] Every FO additively definable binary quantifier is PDA-computable.

This means that the proportional quantifiers such as half or at least two-thirds need the extra computing power of a stack to keep track of the relative number of 1's and 0's. Similarly, we refer to such quantifiers as *(deterministic) context free quantifiers*.

3 New Proof of Closure under Iteration

Shane Steinert-Threlkeld and Thomas Icard III made the first foray into semantic automata for polyadic quantifiers with their paper [23]. First we present a convention for translating models with binary relations into strings. Then we reprove the results that regular and context-free languages are closed under quantifier iteration in a cleaner fashion that makes explicit the intuitions grounding their proofs.

In order to talk about the language accepted by an automaton for an iterated quantifier, we need a way of translating models with *non-unary relations* into strings.[2] The idea is simple: given a binary relation R with domain A and range B, look in turn at every element a of A and record, for each element b of B, whether or not a is in the relation R with b. To keep the substrings generated by each a distinguishable, we introduce a new separator symbol $\boxdot$.

Example 3.1. A quick example will make the definition to follow more intuitive. Figure 1 depicts a model with sets A and B and a relation R between the two. This could represent, for example, a set of aunts, a set of books, and the information regarding which aunts read which books. To translate this model into a string, we look at the elements of A in some order (the indices yield a natural enumeration) and examine which elements of B they connect to: a_1 R's every element of B, so we write $111\boxdot$; a_2 R's only the first element, so we write $100\boxdot$; a_3 R's the last two elements, so we write $011\boxdot$. Concatenating these three substrings yields $111\boxdot100\boxdot011\boxdot$, which is the string representation of the model.

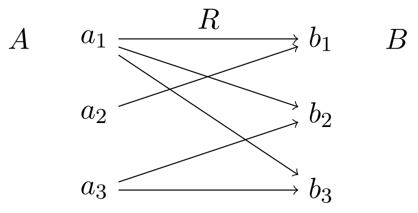

Figure 1: Example for Definition 3.2

Definition 3.2. Let $\mathcal{M} = \langle M, A, B, R\rangle$ be a model, $\vec{a}$ and $\vec{b}$ enumerations of A and B, and let $n =\mid A \mid$. Define a new translation function τ_2 which takes two sets and a binary relation as arguments:

$$\tau_2(\vec{a}, \vec{b}, R) = (\tau(\vec{b}, R_{a_i})\boxdot)_{i\leq n}$$

where $R_{a_i} = \{b \in B : (a_i, b) \in R\}$ is the set of b in B in the relation R with a_i. That is, for each a_i, τ computes a substring with a separator symbol $\boxdot$ appended to the end, recording a 1 if b_j is in R_{a_i} and a 0 otherwise. The final string is the concatenation of all these substrings.

Now languages of iterated type $\langle 1, 1, 2\rangle$ quantifiers are straightforward extensions of languages in the monadic type $\langle 1, 1\rangle$ case. Recall that a quantifier Q_1 is equivalently a binary relation Q_1^c between the number of 1's and 0's in the strings of its language. For quantifiers of the form $\mathsf{Q}_1 \cdot \mathsf{Q}_2$, we let subwords (sequences of 1's and 0's separated by $\boxdot$'s) in the language of Q_2 replace 1's and subwords in the complement of the language of Q_2 replace 0's as the units upon which Q_1^c is defined. Whether or not a subword is in the language of Q_2 is just an instance of the simple monadic case.

To see that this is the correct intuition, recall the def-

[2]For simplicity, in this paper, we assume the relations are binary, but all the results may be easily generalized [17].

inition of binary iterations:

$$(\mathsf{Q}_1 \cdot \mathsf{Q}_2)(A, B, R) \Leftrightarrow \mathsf{Q}_1(A, \{a : \mathsf{Q}_2(B, R_a)\})$$

If we denote the set $\{a : \mathsf{Q}_2(B, R_a)\}$ by X, then a single a (yielding a 1 by τ) is in $A \cap X$ if and only if Q_2-many b are in $B \cap R_a$ (yielding a string in the language of Q_2 by τ). Thus we also have that a is in $A - X$ (yielding a 0) if and only if it's not the case that Q_2-many b are in $B \cap R_a$ (yielding a string not in the language of Q_2–equivalently, a string in the complement). Since they are equivalent, we write $w \notin \mathcal{L}_{\mathsf{Q}_2}$ and $w \in \mathcal{L}_{\neg\mathsf{Q}_2}$ interchangeably throughout.

Definition 3.3. Let Q_1 and Q_2 be quantifiers of type $\langle 1, 1\rangle$. We define the language of $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ by

$$\mathcal{L}_{\mathsf{Q}_1 \cdot \mathsf{Q}_2} = \{w \in (w_i \boxdot)^* : w_i \in \{0,1\}^* \text{and} \\ (\mid \{w_i : w_i \notin \mathcal{L}_{Q_2}\} \mid, \mid \{w_i : w_i \in \mathcal{L}_{\mathsf{Q}_2}\} \mid) \in \mathsf{Q}_1{}^c\}.$$

Example 3.4. The language of the iterated quantifier some · every still ultimately reduces to a numerical constraint on the number of 1's and 0's in strings of the language:

$$\begin{aligned} s \in \mathcal{L}_{\mathsf{some} \cdot \mathsf{every}} &\Leftrightarrow \\ (\mid \{w_i : w_i \notin \mathcal{L}_{\mathsf{every}}\} \mid, \mid \{w_i : w_i \in \mathcal{L}_{\mathsf{every}}\} \mid) \\ \in \mathsf{some}^c &\Leftrightarrow \mid \{w_i : w_i \in \mathcal{L}_{\mathsf{every}}\} \mid > 0 \\ &\Leftrightarrow \mid \{w_i : (\#_0(w_i), \#_1(w_i)) \in \mathsf{every}^c\} \mid > 0 \\ &\Leftrightarrow \mid \{w_i : \#_0(w_i) = 0\} \mid > 0 \end{aligned}$$

By a similar derivation we get:

$$s \in \mathcal{L}_{\mathsf{every} \cdot \mathsf{some}} \Leftrightarrow \mid \{w_i : \#_1(w_i) = 0\} \mid = 0$$

The string from Example 3.1, $111 \boxdot 100 \boxdot 011 \boxdot$, is a member of both these languages, indicating that the sentences *Every A R some B* and *Some A R every B* are both true in the model depicted by Figure 1.

In the following sections we often speak of *words in the language of* Q_2 without explicitly stating whether we mean words in $\{0,1\}$ or words ending in $\boxdot$. When considering these strings as input for iteration automata, we will make reference to the *well-formedness* of a string. Call a string *well-formed* if it ends in $\boxdot$. The reader may wonder why we don't define well-formedness in terms of individual subwords. To explain this, we must point out that the language accepted by an iteration automaton is in a sense bigger than the number of relations whose translation it accepts [17][3].

Steinert-Threlkeld and Icard III argue the closure of regular and context-free languages under quantifier iteration via arguments from regular expressions and context-free grammars, respectively. Their intuitive justification does not outright mention the general closure of regular and context-free languages under substitution; however, it is informative to deliberately state the fact that quantifier iteration *just is* an instance of substitution, from which these closure results follow straightforwardly. In our reformulations of their proofs, we define substitutions directly on languages.

Theorem 3.5. Let $\mathcal{L}_{\mathsf{Q}_1}$ and $\mathcal{L}_{\mathsf{Q}_2}$ be languages of type $\langle 1, 1\rangle$ regular quantifiers with alphabets $\Sigma_1 = \Sigma_2 = \{0,1\}$. $\mathcal{L}_{\mathsf{Q}_1 \cdot \mathsf{Q}_2}$ is a regular language.

Proof. Define a substitution s on $\mathcal{L}_{\mathsf{Q}_1}$ by the following:

- $s(0) = \mathcal{L}_{\neg\mathsf{Q}_2}\boxdot$
- $s(1) = \mathcal{L}_{\mathsf{Q}_2}\boxdot$

<u>Claim:</u> $s(\mathcal{L}_{\mathsf{Q}_1}) = \mathcal{L}_{\mathsf{Q}_1 \cdot \mathsf{Q}_2}$

<u>Proof:</u> This is immediately clear from the substitution. For $w = (w_i \boxdot)^*$, $w \in \mathcal{L}_{\mathsf{Q}_1 \cdot \mathsf{Q}_2}$ if and only if $(\mid \{w_i : w_i \in \mathcal{L}_{\neg\mathsf{Q}_2}\} \mid, \mid \{w_i : w_i \in \mathcal{L}_{\mathsf{Q}_2}\} \mid) \in \mathsf{Q}_1^c$, if and only if $w = s(w')$ where $(\#_0(w'), \#_1(w')) \in \mathsf{Q}_1^c$, if and only if $w \in s(\mathcal{L}_{\mathsf{Q}_1})$. ■

Thus s is the appropriate substitution. Since regular languages are closed under complement, $\mathcal{L}_{\neg\mathsf{Q}_2}$ is regular, and since regular languages are closed under concatenation, $\mathcal{L}_{(\neg)\mathsf{Q}_2}\boxdot$ is regular. Thus s defines a regular substitution, so by regular substitution closure, $s(\mathcal{L}_{\mathsf{Q}_1}) = \mathcal{L}_{\mathsf{Q}_1 \cdot \mathsf{Q}_2}$ is a regular language. □

Theorem 3.6. Let $\mathcal{L}_{\mathsf{Q}_1}$ and $\mathcal{L}_{\mathsf{Q}_2}$ be languages of type $\langle 1, 1\rangle$ context-free quantifiers with alphabets $\Sigma_1 = \Sigma_2 = \{0,1\}$. $\mathcal{L}_{\mathsf{Q}_1 \cdot \mathsf{Q}_2}$ is a context-free language.

Proof. We use the same substitution s on $\mathcal{L}_{\mathsf{Q}_1}$:

[3] For a model $\mathcal{M} = (M, A, B, R)$ with $n = \mid A \mid$ and $m = \mid B \mid$, τ_2 generates strings of the form $((1+0)^m \boxdot)^n$, but $\mathcal{L}_{\mathsf{Q}_1 \cdot \mathsf{Q}_2}$ contains strings of the form $((1+0)^* \boxdot)^*$, where subwords need not have equal length.

- $s(0) = \mathcal{L}_{\neg \mathsf{Q}_2}\boxdot$
- $s(1) = \mathcal{L}_{\mathsf{Q}_2}\boxdot$

Claim: $s(\mathcal{L}_{\mathsf{Q}_1}) = \mathcal{L}_{\mathsf{Q}_1 \cdot \mathsf{Q}_2}$

Proof: The argument for the previous Claim holds here as well. ■

Since context-free *quantifier* languages are closed under complement[4], $\mathcal{L}_{\neg \mathsf{Q}_2}$ is context-free, and since context-free languages are closed under concatenation, $\mathcal{L}_{(\neg)\mathsf{Q}_2}$ is context-free. Thus s defines a context-free substitution, so by context-free substitution closure, $s(\mathcal{L}_{\mathsf{Q}_1}) = \mathcal{L}_{\mathsf{Q}_1 \cdot \mathsf{Q}_2}$ is a context-free language.

□

4 Constructing minimal iteration DFA

As mentioned previously, the construction in [23] is overly powerful, creating a pushdown automaton as the iteration of two DFA. Their construction of $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ consists of a copy of Q_2 that pushes a 1 (0) to the stack for every subword in $\mathcal{L}_{\mathsf{Q}_2}$ ($\mathcal{L}_{\neg \mathsf{Q}_2}$) and a "pushdown reader" Q_1^P that "reads" the resulting stack. Further, in that paper they state "There appears to be no such analogously general mechanism for generating minimal DFAs." Of course, there is great theoretical and practical interest in identifying the least-powerful automata recognizing iterated quantifiers. The duality between languages and automata makes formal language theory interesting in its own right, and the fact that automata often represent intuitive algorithms for string-membership provides further motivation from the perspective of modeling quantifier verification. We show that, as the iteration closure of regular languages suggests, there is a general method to construct an iteration DFA from two DFA, and furthermore we can directly construct the near-minimal version in every case.

The definition of languages of iterated quantifiers already suggests how to go about constructing iterated automata from the monadic building blocks. For languages, just replace 1's in the first by entire words in the language of the second, and 0's by entire words in the complement of the language of the second. To complete the picture, we must ask ourselves, *what is the analogous notion in terms of automata*? Quite simply, *1-transitions* of the first automaton should be replaced by *accepting runs* of the second automaton and *0-transitions* replaced by *rejecting runs.*

The main idea is to start with Q_1 as the backbone of $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ and then replace each of its states with a copy of Q_2. To make things easier, imagine these copies are indexed by the state they replace. From here on we refer to such copies by Q_2^q and refer to their components in the obvious way (e.g. $\mathcal{Q}_2^q, s_2^q, \delta_2^q, F_2^q$). If some copy Q_2^q ends in a final state seeing some subword, then the machine should behave as if Q_1 had seen a 1. Suppose q would transition to p on a 1. This means that every final state of Q_2^q should transition to the start state of Q_2^p on $\boxdot$ (as this marks the end of the subword). Similarly, every rejecting state of Q_2^q should have a $\boxdot$-transition to Q_2^r, where r is the state that q would transition to on a 0. $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ has the same start and final states as Q_1.

Before giving the formal definition, let us dig a little deeper with an example. Consider the automaton $\mathsf{exactly\ three} \cdot \mathsf{every}$ depicted in Figure 3. The vestiges of the original state-space of $\mathsf{exactly\ three}$ is clearly visible as the "spine" of the automaton, enclosed in the darker dashed box, but the original states have been replaced by copies of every, enclosed in the lighter dashed boxes. There are three exceptions to this simple replacement scheme:

(i) Final states: Notice that the final state of $\mathsf{exactly\ three}$ remains externally linked up with a copy of every. This is so that the automaton cannot erroneously accept a word ending in 111, for example, which is not well-formed.

(ii) Terminal states: Notice that the terminal state of $\mathsf{exactly\ three}$ doesn't seem to have been replaced at all. If the automaton reaches state q_5, the rest of the input is irrelevant. The automaton can *only* reject at this point, hence the looping on every symbol.

(iii) Final, terminal states: The current example does not exhibit this exception, but when an au-

[4]This follows from Theorem 2.11, since semilinear or FO definable sets are closed under complement.

tomaton reaches a state that is both final and terminal, it should accept irrespective of the remaining input *so long as* it is well-formed. Such states q require at most one extra state e_q to go to in which to loop on 0 and 1 and then return to q on ⧈. If q has a predecessor state r with equivalent behavior, then the extra state is unnecessary (as q may just go to r on 0 and 1).

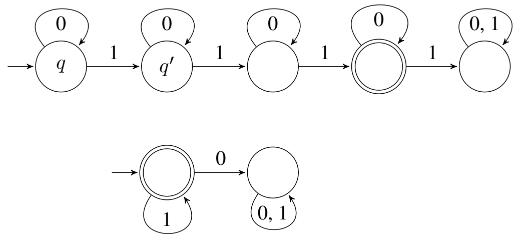

Figure 2: exactly three and every

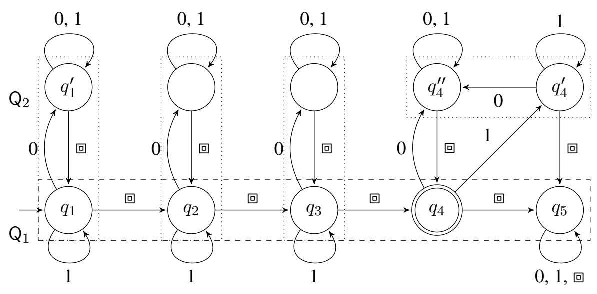

Figure 3: exactly three·every

As the state-space is given by the replacement scheme, and the 1 and 0-transitions are given by the copies of Q_2, all that remains is to specify the ⧈-transitions, which are determined by the 1 and 0-transitions in the original Q_1. Consider the states q_1 and q_1', together comprising Q_2^q, replacing q in exactly three. Since q_1 is an accepting state in every, a subrun ending there is analogous to a 1. Thus the ⧈-transition from q_1 should mimic the 1-transition of q to q', going to the start state of the copy of Q_2 replacing q'. Similarly, q_1' should mimic the 0-behavior of q, which is to loop, meaning q_1' should return to the start state of the Q_2 copy of which it is a member.

The final state q_4 is not itself a member of any copy of Q_2, but its ⧈ transitions are still decided by the start state of its associated copy of Q_2. If ⧈ is seen in such a state, this means the current subword was empty; this works because ϵ (the empty string) is in the language of Q_2 if and only if s_2 is final, so s_2 appropriately determines ⧈-behavior.

Definition 4.1. Previously we likened the state space of Q_1 to the spine or essential structure of $\mathsf{Q}_1 \cdot \mathsf{Q}_2$. Here we make this idea precise by describing a mapping between $\mathcal{Q}_1$ and a subspace of $\mathcal{Q}$ (the state-space of the iteration automaton). Define a bijection $f : \mathcal{Q}_1 \to \mathcal{Q}$ by the following:

$$f(q) = \begin{cases} q_F & q \in F_1, \text{ non-terminal} \\ s_2^q & q \notin F_1, \text{ non-terminal} \\ q_{FT} & q \in F_1, \text{ terminal} \\ q_T & q \notin F_1, \text{ terminal} \end{cases}$$

For example, if a state in $\mathcal{Q}$ has the subscript F, then the corresponding state in $\mathcal{Q}_1$ must have been both final and non-terminal. Not every s_2^q in $\mathcal{Q}$ is $f(q)$ for some $q \in \mathcal{Q}_1$, but if $q \in \mathcal{Q}_1$ is non-final and non-terminal, it will be merged with the start state of its copy of Q_2. This state mapping will be mostly useful for defining the transition function for the iteration automata. Using this mapping to convert between q and $f(q)$ and vice versa, we can be sensitive to the above-mentioned exceptions in the replacement scheme while still using δ_1 to define δ. When the value of f for some q is clear, we may write it directly, e.g. "q_{FT}" in lieu of "$f(q)$," and similarly for f^{-1}. Sometimes we write, e.g., q_T to mean a specific state, and other times to mean the set of all states that are $f(q)$ for some non-final, terminal state q. The intention should be clear from the context.

Definition 4.2. Let $\mathsf{Q}_1 = (\mathcal{Q}_1, \Sigma_1, \delta_1, s_1, F_1)$ and $\mathsf{Q}_2 = (\mathcal{Q}_2, \Sigma_2, \delta_2, s_2, F_2)$ be DFAs accepting the monadic quantifier languages $\mathcal{L}_{\mathsf{Q}_1}$ and $\mathcal{L}_{\mathsf{Q}_2}$, respectively. The iteration DFA $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ is given by:

- $\mathcal{Q}$: $\displaystyle\bigcup_{q \in \mathcal{Q}_1} \begin{cases} \mathcal{Q}_2^q \cup \{q_F\} & q \in F_1, \text{ non-terminal} \\ \mathcal{Q}_2^q & q \notin F_1, \text{ non-terminal} \\ \{e_q, q_{FT}\} & q \in F_1, \text{ terminal} \\ \{q_T\} & q \notin F_1, \text{ terminal} \end{cases}$

  Here e_q is the (potentially unnecessary) state added to make sure input seen in q_{FT} is well-formed.

- $\Sigma = \{0, 1, ⧈\}$

- Transition function:

  - For $p \in \mathcal{Q}_2^q : \delta(p, x) = \begin{cases} \delta_2^q(p, x) & x \in \{0,1\} \\ f(\delta_1(f^{-1}(p), 1)) & x = \boxdot, p \in F_2^q \\ f(\delta_1(f^{-1}(p), 0)) & x = \boxdot, p \notin F_2^q \end{cases}$
  - For $q \in \{q_F\}$: $\delta(q, x) = \delta_2^q(s_2^q, x)$
  - For $q \in \{q_T\}$: $\delta(q, x) = q$
  - For $q \in \{q_{FT}\}$: $\delta(q, x) = \begin{cases} e_q & x \in \{0,1\} \\ q & x = \boxdot \end{cases}$
  - For $p \in \{e_q\}$: $\delta(p, x) = \begin{cases} e_q & x \in \{0,1\} \\ q & x = \boxdot \end{cases}$
- $s = f(s_1)$
- $F = \{f(q) \mid q \in F_1\}$

As remarked earlier, the e_q may not be necessary, but whether they are needed is easily seen after δ has been specified. Once the above construction is completed, one must inspect the state p such that $\delta(p, \boxdot) = q$, for each q in q_{FT}. If p also loops on 0 and 1, then e_q can be removed, and δ amended such that q transitions to p on $\boxdot$.

The definition of $\mathcal{Q}$ makes obvious the following upper bound on the size of iteration DFAs:

Fact 4.3. The state space of $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ is at most

$$\sum_{q \in \mathcal{Q}_1} \begin{cases} |\ \mathcal{Q}_2\ | + 1 & q \in F_1, \text{ non-terminal} \\ |\ \mathcal{Q}_2\ | & q \notin F_1, \text{ non-terminal} \\ 2 & q \in F_1, \text{ terminal} \\ 1 & q \in F_1, \text{ non-terminal} \end{cases}$$

This fact gives us the *state complexity* of iteration DFA, which is a worst-case notion of state-space size[5], and thus an upper bound.[6] Since our definition uses cases, it is generally tight for any language. Iteration DFA may have fewer states if, as discussed above, extra states e_q are not needed to ensure well-formedness of the input (see Figure 4). Thus, the size of $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ is generally within $m = |\ \{q_{FT}\}\ |$ of this upper bound (and m is at most 1 for regular quantifiers). It is also possible for unforseen state equivalences to occur (see, for example, Figure 5), but such minimizations are similarly restricted to "end behavior."

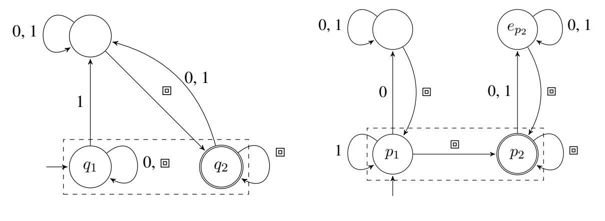

Figure 4: In (a), the terminal final state q_2 of the outer some can $0, 1$-transition to the terminal state of the embedded some. In (b), our definition correctly predicts the necessity of e_{p_2}.

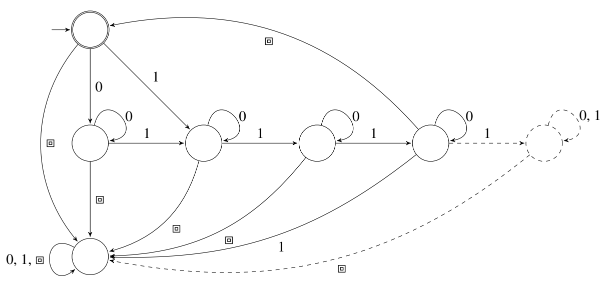

Figure 5: every·exactly three. Solid lines indicate the minimal DFA. Dashed lines indicate the output of the construction, with a full copy of exactly three.

We now make explicit the correctness of the definition of iteration automata by showing that the language accepted by the automaton constructed from the automata for regular quantifiers Q_1 and Q_2 and the language $\mathcal{L}_{\mathsf{Q}_1 \cdot \mathsf{Q}_2}$ are equivalent. To prove this, we first introduce a bit more helpful terminology and a preliminary lemma. Define a function $g : \{0,1\}^* \to \{0,1\}$ by:

[5]See [8] for an overview of the concept. "The *state complexity* of a regular language L . . . is the number of states of its minimal DFA," and "[t]he *state complexity of an operation* (or *operational state complexity*) on regular languages is the worst-case complexity of a language resulting from the operation, considered as a function of the state complexities of the operands."

[6]In [22] (2014), Steinert-Threlkeld also gives a definition (developed independently) of iteration DFA for type $\langle 1, 1, 2 \rangle$ regular iterations (different to the PDA construction developed in [23]). His definition uses the cross-product of $\mathcal{Q}_1$ and $\mathcal{Q}_2$ for the state space with an "unrolled" version of Q_2 with an extra state if s_2 is final. This leads to a state complexity of $|\ \mathcal{Q}_1\ | \cdot |\ \mathcal{Q}_2 + 1\ |$. Using the function f to distinguish different cases in defining the state space, we achieve a smaller state complexity that only reaches $|\ \mathcal{Q}_1\ | \cdot |\ \mathcal{Q}_2 + 1\ |$ in case Q_1 is a trivial language (having only final states).

$$g(w) = \begin{cases} 0 & w \notin \mathcal{L}_{\mathsf{Q}_2} \\ 1 & w \in \mathcal{L}_{\mathsf{Q}_2} \end{cases}$$

so g is the characteristic function of $\mathcal{L}_{\mathsf{Q}_2}$, and let $g'(w) = g(w_i)_{i \le \#_{\boxdot}(w)}$, where $w \in (w_i\boxdot)^*$. For example, letting Q_2 = every, we can calculate $g'(111\boxdot 101\boxdot) = g(111)g(101) = 10$.

Using g and the function f from Definition 4.1, in the following lemma we prove the intuition grounding our construction in the first place: that transitions on words in the language of Q_2 and its complement in $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ are somehow equivalent to transitions on 1's and 0's in Q_1. See Figure 6 for an illustration of this idea. Given this correspondence, the desired result will be easy to see.

Lemma 4.4. For $w_i \in \{0,1\}$ and $p = f(q)$ for some $q \in \mathcal{Q}_1$, $\delta(p, w_i\boxdot) = f(\delta_1(f^{-1}(p), g(w_i)))$. This means that the state $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ reaches from p reading $w_i\boxdot$ is the result of applying f to the state that Q_1 reaches from $f^{-1}(p)$ reading $g(w_i)$.

$$\begin{tikzcd}
p \arrow[r, bend left, "\delta(p, w_i\boxdot)"] \arrow[d, "f^{-1}"'] & f(q) \\
f^{-1}(p) \arrow[r, bend left, "\delta_1(f^{-1}(p), g(w_i))"] & q \arrow[u, "f"']
\end{tikzcd}$$

Figure 6: Diagram for Lemma 4.4

Proof. There are four cases to consider, depending on what kind of state p is:

(i) s_2^q: Suppose $w_i \in \mathcal{L}_{\mathsf{Q}_2}$. Then $\delta(s_2^q, w_i) = p$ where $p \in F_2^q$, and $\delta(p, \boxdot) = f(\delta_1(q, 1))$, which is precisely $f(\delta_q(f^{-1}(s_2^q), g(w_i)))$. The case for $w_i \notin \mathcal{L}_2$ is symmetric.

(ii) q_F: Suppose $w_i \neq \epsilon$, so $w_i = xw_i'$ where $x \in \{0,1\}$. Then $\delta(q_F, x) \in \mathcal{Q}_2^q$, and this collapses to case (i). Suppose $w_i = \epsilon$, and $\epsilon \in \mathcal{L}_{\mathsf{Q}_2}$, so $g(w_i) = 1$. Then $\delta(q_F, \boxdot) = \delta(s_2^q, \boxdot) = f(\delta_1(q, 1))$, since $s_2^q \in F_2^q$ (and similarly if $\epsilon \notin \mathcal{L}_{\mathsf{Q}_2}$).

(iii) q_{FT}: Since $q = f^{-1}(q_{FT})$ is terminal, $\delta_1(q, g(w_i)) = q$. If $w_i = \epsilon$, then $\delta(q_{FT}, \boxdot) = q_{FT} = f(q)$. If not, then $\delta(q_{FT}, w_i) = e_q$, and $\delta(e_q, \boxdot) = q_{FT} = f(q)$.

(iv) q_T: Again, $q = f^{-1}(q_T)$ is terminal, so $\delta_1(q, g(w_i)) = q$, and $\delta(q_T, w_i\boxdot) = q_T = f(q)$.

□

Theorem 4.5. The language accepted by $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ is $\mathcal{L}_{\mathsf{Q}_1 \cdot \mathsf{Q}_2}$.

Proof. It follows from Lemma 4.4 that if w is a string of the form $(w_i\boxdot)^*$, $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ accepts w visiting a sequence of states $s_2^{s_1} q_1 \cdots q_n$ (with $q_i = f(q)$ for some $q \in \mathcal{Q}_1$, and possibly repeating) if and only if Q_1 accepts the string $g'(w)$ visiting the sequence of states $s_1 f^{-1}(q_1) \cdots f^{-1}(q_n)$. That is, $w \in L(\mathsf{Q}_1 \cdot \mathsf{Q}_2)$ if and only if $g'(w) \in \mathcal{L}_{\mathsf{Q}_1}$. But $g'(w) \in \mathcal{L}_{\mathsf{Q}_1}$ if and only if $(\#_0(g(w)), \#_1(g(w))) \in \mathcal{Q}_1^c$, if and only if, by definition, $(| \{w_i : w_i \notin \mathcal{L}_{\mathsf{Q}_2}\} |, | \{w_i : w_i \in \mathcal{L}_{\mathsf{Q}_2}\} |) \in \mathsf{Q}_1^c$, which is the definition of membership for $\mathcal{L}_{\mathsf{Q}_1 \cdot \mathsf{Q}_2}$. □

5 Closure of DCFLs under Iteration

Now, it is natural and relevant to ask whether *deterministic* context-free quantifier languages, identified recently by Kanazawa [11], are closed under iteration. In this section we answer this open question in the affirmative.[7] The result is not obvious since DCFLs do not enjoy general substitution closure.

First we establish a sort of normal form for DPDA that is necessary to preserve determinism in the resulting iteration automaton.

Lemma 5.1. For every DPDA P recognizing some $\mathcal{L}_{\mathsf{Q}}$ that is the language of a deterministic context-free type $\langle 1,1 \rangle$ quantifier Q, there is a DPDA P' with the following properties:

[7] This closure result was announced independently by Shane Steinert-Threlkeld and a proof sketch appears in [22], however, it indicates that the DPDA construction proceeds similarly to the DFA case. Since simply complementing the accepting states of a given DPDA may not result in the correct behavior (as it may continue to transition between accepting and rejecting states after reading the input), the correctness of our definitions in this section relies on the modifications described in Lemma 5.1. The cases are not necessarily similar (that is, we can not necessarily use a single DPDA Q_2 to decide if $w \in L_{\mathsf{Q}_2}$ or $w \in L_{\neg \mathsf{Q}_2}$) without the kind of normal form for DPDA we describe here.

1. P' has a single accept state q_{accept} such that $(q_0, w⊡, \epsilon) \overset{*}{\vdash} (q_{\text{accept}}, \epsilon)$ if and only if $w \in \mathcal{L}_{\mathsf{Q}}$

2. P' has a state q_{reject} such that $(q_0, w⊡, \epsilon) \overset{*}{\vdash} (q_{\text{reject}}, \epsilon)$ if and only if $w \in \mathcal{L}_{\neg\mathsf{Q}}$

That is, given P recognizing the language of Q, we can construct another DPDA that in a sense recognizes both Q and ¬Q by empty stack given an endmarker.

Proof. This follows from the complementation closure of DCFL and the ability to recognize the end of the string. The original DPDA may enter both final and non-final states via ϵ moves after the last symbol, so inverting final and non-final states is insufficient to construct the complement. The key is to identify a set of reading states R without ϵ moves and a new set of final states F contained in R such that $R - F$ is accepting for the complement of P. Then add a new accept state q_{accept} and modify the transition function such that the automaton empties its stack and goes to q_{accept} if it enters a state in F after reading ⊡, satisfying (1). We do the same for a new state q_{reject} and $R - F$, satisfying (2). □

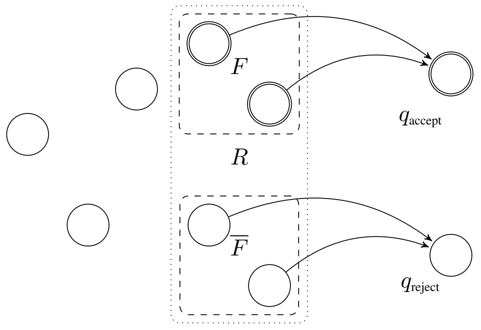

Figure 7: End result of DPDA modification according to Lemma 5.1

Now we can demonstrate the following with a proof by automata[8]:

Theorem 5.2. Deterministic context-free languages are closed under quantifier iteration.[9]

[8]An alternative proof via iterated grammars and the DK-test (see [15] and [21]) is also presented in [17]

[9]Note that, though we state this proof in terms of quantifier languages, it applies to the "quantifier iteration" of any two binary DPDA-recognizable languages.

Recall that we use the notation $\langle q, x, \alpha, \beta, q'\rangle$ for $\delta(q, x, \alpha) = (q', \beta)$ for (D)PDA.

Definition 5.3. Let $\mathsf{Q}_1 = (\mathcal{Q}_1, \Sigma_1, \Gamma_1, \delta_1, s_1, F_1)$ be any DPDA recognizing a deterministic context-free quantifier language $\mathcal{L}_{\mathsf{Q}_1}$. Let $\mathsf{Q}_2 = (\mathcal{Q}_2, \Sigma_2, \Gamma_2, \delta_2, s_2, q_{\text{accept}}, q_{\text{reject}})$ be a DPDA modified according to Lemma 5.1 recognizing an endmarked deterministic context-free quantifier language $\mathcal{L}_{\mathsf{Q}_2}$. Define the iteration DPDA $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ by:

- $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$
- $\Sigma = \Sigma_1 \cup \Sigma_2$
- $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \mathcal{Q}_1$
- $\delta =$

$$\begin{aligned}
&\delta_2 && (1)\\
&\cup\{\langle q, \epsilon, \alpha, \beta, q'\rangle : \langle q, \epsilon, \alpha, \beta, q'\rangle \in \delta_1\} && (2)\\
&\cup\{\langle q, \epsilon, \alpha, q\alpha, s_2\rangle : (q, x, \alpha) \in \text{dom}(\delta_1) && (3)\\
&\qquad\qquad \text{and } x \in \{0,1\}\\
&\cup\{\langle q_{\text{accept}}, \epsilon, q\alpha, \beta, q'\rangle : \langle q, 1, \alpha, \beta, q'\rangle \in \delta_1\} && (4)\\
&\cup\{\langle q_{\text{reject}}, \epsilon, q\alpha, \beta, q'\rangle : \langle q, 0, \alpha, \beta, q'\rangle \in \delta_1\} && (5)
\end{aligned}$$

- $s = s_1$
- $F = F_1$

We take the states of Q_1 and the states of Q_2 and connect them in the following way: for every transition in δ_1 in which some state q reads a symbol, we replace that transition with an ϵ transition to the start state of Q_2 and push q to the stack. Thus all subwords $w_i⊡$ of the input are processed by Q_2; in any case, Q_2 empties its stack up to q and ends up in one of q_{accept} or q_{reject}, and transitions back into Q_1—with the new state and new stack contents decided by q.

Of course, natural language iterations often involve a mixture of regular and context-free quantifiers:

(i) *A third of the students answered every question correctly.*

(ii) *Fewer than five students attended more than half of the presentations.*

Since REG⊂DCFL, every DFA can be converted to an equivalent DPDA with no stack manipulation, so the above definition is general enough to accommodate the case that one or both of the input automata are DFA.

Claim 5.4. The automaton $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ yielded by Definition 5.3 is deterministic.

Proof. First we show this holds when both Q_1 and Q_2 are DPDA. We show there is only one move per configuration in δ by examining each part (1)-(5) of the definition:

(1) δ_2 has at most one move per configuration.

(2) δ_1 has at most one move per configuration.

(3) A transition of this type is added if q has 0,1 moves in δ_1 with α on the stack. This means q does not have an ϵ move with α on the stack in δ_1 (or a transition with both ϵ input and stack). Thus, replacing 0,1 with ϵ with α on the stack leaves q with one choice in δ.

(4) In δ_2, q_{accept} has no moves by construction, and δ_1 is deterministic, so there is exactly one move in δ for configuration $(p, \epsilon, q\alpha)$.

(5) The same argument in (4) applies for q_{reject}.

$\square$

To see the correctness of this definition, we again prove a lemma relating transitions on $\boxdot$-ended words in $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ to transitions on individual symbols in Q_1.

Lemma 5.5. Let g be the characteristic function of $\mathcal{L}_{\mathsf{Q}_2}$. For $w_i \in \{0,1\}^*$ and $q \in \mathcal{Q}_1$, $\delta(q, w_i\boxdot, \alpha) = \delta_1(q, g(w_i), \alpha)$.

Proof. Let $\mathsf{Q}_1, \mathsf{Q}_2$ both be DPDA. Assume w.l.o.g. that q has 0,1-transitions in δ_1 (otherwise there is an ϵ-transition to some q', in both δ_1 and δ_2, with the same effect on the stack (2)). Then in δ, q has an ϵ-move to s_2 with q pushed to the stack (3). Since q is not in Γ_2, this is effectively an empty stack to δ_2, so by (1) and Lemma 5.1 we have that $\delta(s_2, w_i\boxdot, q\alpha)$ goes to $(q_{\text{accept}}, q\alpha)$ if $g(w_i) = 1$ or $(q_{\text{reject}}, q\alpha)$ if $g(w_i) = 0$. By (4) and (5), there is an ϵ-move to $\delta_1(q, g(w_i), \alpha)$. $\square$

Theorem 5.6. The language accepted by the DPDA $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ is $\mathcal{L}_{\mathsf{Q}_1 \cdot \mathsf{Q}_2}$.

Proof. Given the above lemma, the proof is very similar to that of Theorem 4.5 $\square$

6 Frege Boundary

Iterations represent a kind of default, the 'bread and butter' of multiple quantification in natural language, hence a popular proposal, so-called Frege's Thesis: *All polyadic quantification in natural language is iterated monadic quantification.* The Frege boundary demarcates the line between reducible and irreducible polyadic quantifiers. Historically, when proposing the boundary, Van Benthem [3] referred to Frege, who introduced the familiar notion of quantification to modern logic. Frege was also the first to give a satisfactory analysis of multiple quantification, by simply taking every instance of multiple quantification to be an iteration. Van Benthem calls this 'solving the problem by ignoring it'—since within this view we can preemptively give an account of any polyadic quantifier in terms of simple monadic quantifiers. Thus, those polyadic quantifiers that can be analyzed as iterations of monadic quantifiers are deemed *reducible*, or simply *Fregean*. Those that can be given no such analysis are *irreducible* or *non-Fregean*, and may be considered *genuinely polyadic*.

$(\mathsf{Q}_1 \cdot \mathsf{Q}_2)(A, B, R)$ is simply $\mathsf{Q}_1 a \mathsf{Q}_2 b R(a, b)$, and thus iteration is *monadically definable*: this is the sense in which the lift is not taken to be genuinely polyadic. The other lifts, for instance, cumulation and constructions containing *same* and *different* are generally not reducible to iterations. But how we can characterize the Frege boundary? What makes a quantifier non-Fregean?

6.1 Classic Characterization Results

Let us start with a definition to systematize the above discussion:

Definition 6.1. Let us call a type $\langle 2 \rangle$ quantifier *Fregean* if it is an iteration of monadic quantifiers (or a Boolean combination thereof). We say a quantifier 'lies beyond the Frege boundary' if it is not Fregean.

We proceed historically, starting with the first characterization:

Theorem 6.2 ([3])**.** On any finite universe, a binary quantifier Q is a *right complex* (a Boolean combination of iterations) if and only if it is both logical and right-oriented.

A quantifier is logical if it is closed under permutations of individuals: $R \in \mathsf{Q}$ if and only if any $\pi(R) \in \mathsf{Q}$. If $S = \pi(R)$, we write $S \approx R$, and say that Q is closed under $\approx$. A quantifier is right-oriented if it is closed under $\sim$, where we write $R \sim S$ if for all x, $|(R_x)| = |(S_x)|$. This corresponds to preserving the entire arrow pattern of a relation and preserving the outgoing arrow pattern of a relation.[10]

[14] provides a characterization that also applies to nonlogical quantifiers and relies on the interesting observation that if two reducible quantifiers behave the same on relations that are cross-products, they actually behave the same on every relation (i.e., are equivalent).

Theorem 6.3 ([14])**.** For reducible type (2) quantifiers Q and Q′, $\mathsf{Q} = \mathsf{Q}'$ if and only if for all subsets A, B of M, $\mathsf{Q}(A \times B) = \mathsf{Q}'(A \times B)$.

The following equivalent statement of the theorem provides a test for reducibility: if $\mathsf{Q}(A \times B) = \mathsf{Q}'(A \times B)$ for all $A, B \in \mathcal{P}(M)$, and we know $\mathsf{Q}' = \mathsf{Q}_1 \cdot \mathsf{Q}_2$, then Q is reducible if and only if $\mathsf{Q} = \mathsf{Q}_1 \cdot \mathsf{Q}_2$.

Dekker then generalizes this to quantifiers of arbitrary arity:

Theorem 6.4 ([6])**.** For type (n) quantifiers Q and Q′ that are n-reducible, $\mathsf{Q} = \mathsf{Q}'$ if and only if for all subsets $A_1, \ldots, A_n$ of M, $\mathsf{Q}(A_1 \times \cdots \times A_n) = \mathsf{Q}'(A_1 \times \cdots \times A_n)$.

Therefore, Q and Q′ have the same behavior on cross-products and Q′ is reducible, thus Q is reducible only if it equals Q′.

Dekker also defines Q to be *invariant for sets in products* if $\mathsf{Q}(A_1 \times \cdots \times A_n)$ and $\mathsf{Q}(A_1' \times \cdots \times A_n')$ imply $\mathsf{Q}(A_1 \times \cdots \times A_i' \times \cdots \times A_n)$ and shows Q is invariant for sets if and only if it is product equivalent to some $\mathsf{Q}' = \mathsf{Q}_1 \circ \cdots \circ \mathsf{Q}_n$.[11] Furthermore, the proof actually constructs the Q_i, widening the applicability of the Keenan-style reducibility test by removing the problem that 'maybe one has not tried hard enough' to find the product-equivalent iteration for comparison.

Example 6.5. Consider the sentence *Every professor wrote the same number of recommendation letters*, formalized as $(\mathsf{every}^P,\ \textit{same number}^L)(W)$. This is product-equivalent to $(\mathsf{every}^P \cdot \mathsf{every}^L)(W)$, since when W is a cross-product relation, every p is always connected to every l, and thus incidentally every p is connected to the same number of l. Since these quantifiers are not the same (take a model in which every p is connected to the same number of l, but $|W_p| < |L|$), $(\mathsf{every},\ \textit{same number})$ is not reducible to *any* two unary quantifiers.

Other examples of non-Fregean quantifiers include:

- Reflexives (The type (2) quantifier consisting of all reflexive binary relations is not Fregean.), e.g.:
    1. Every student is enjoying him/herself.
    2. Every company advertises itself.
- Different/different [14], e.g.:
    1. Different students answered different questions.
    2. Truth-conditions:

$$\forall a \neq b \in \text{students}:\ \text{answered}_{(a)} \neq \text{answered}_{(b)}.$$

- Dependent comparatives [12], e.g.:
    1. A certain number of professors read a much larger number of grad school applications.
    2. Truth-conditions:

$$|\text{dom}(\text{read}) \cap \text{professors}| < |\text{ran}(\text{read}) \cap \text{applications}|.$$

- Branching, resumption, cumulatives, Ramseys (see the following chapter for discussion)

[10]Van Benthem's theorem holds for *local* (on a particular finite universe) definability, but can be used to refute definability on *any* universe [26].

[11]We can not find a natural language example where this is useful, but it is needed to show that the property of a relation being symmetric, which is product-equivalent to no iteration, is not reducible. $A \times A$ and $B \times B$ are symmetric, but neither of $A \times B$ nor $B \times A$ is [6].

Dekker nicely sums up what cross-product characterizations tell us about iterations:

> Not only is this a new and welcome generalization, it also gives some insight into the intimate relation between (n)-reducible type $\langle n \rangle$ quantifiers and n-ary product relations. If type $\langle n \rangle$ quantifier F^n is (n)-reducible…then F^n is satisfied by $\mathcal{Q}_1 \times \cdots \times \mathcal{Q}_n$ iff each composing f_i is satisfied by $\mathcal{Q}_i$ [6].

Jan van Eijck [7] introduces the notion of (m, n)-reducibility, making it possible to say something about polyadic quantifiers of type $(m + n)$ that are not fully $(m + n)$-reducible.

Definition 6.6. Q of type (m, n) is (m, n)-reducible if there are Q_1 and Q_2 of types (m) and (n) such that $\mathsf{Q} = \mathsf{Q}_1 \cdot \mathsf{Q}_2$.

Van Eijck also defines the corresponding notions of reducibility equivalence and invariance for sets in products. The striking consequence of generalizing reducibility is the existence of a diamond property and normal form for quantifiers, meaning reducibility is confluent: if a quantifier reduces to two different iterations, these reducts must have a common further decomposition. If Q of type $(m+n)$ reduces both to $\mathsf{Q}_1 \cdot \mathsf{Q}_2$ (of types (m) and (n)) and to $\mathsf{Q}'_1 \cdot \mathsf{Q}'_2$ (of types (m') and $(m + n - m')$), then there exists Q_3 (of type $(m' - m)$) such that $\mathsf{Q} = \mathsf{Q}_1 \cdot \mathsf{Q}_3 \cdot \mathsf{Q}'_2$.

$$\begin{tikzcd}
& \mathsf{Q} \arrow[dl] \arrow[dr] & \\
\mathsf{Q}_1 \cdot \mathsf{Q}_2 \arrow[dr, dashed] & & \mathsf{Q}'_1 \cdot \mathsf{Q}'_2 \arrow[dl, dashed] \\
& \mathsf{Q}_1 \cdot \mathsf{Q}_3 \cdot \mathsf{Q}'_2 &
\end{tikzcd}$$

Figure 8: Van Eijck's diamond property.

Example 6.7. Consider the sentence *Every teacher assigned different students different problems* analyzed as the type $\langle 3 \rangle$ quantifier (every^T, *different*^{S}, *different*^{P}) applied to the *assign* relation, and let $\mathsf{0}$ denote the unary quantifier that is false of every set. By Dekker's results we can see this is not fully 3-reducible, since it is equivalent to $\mathsf{every} \cdot \mathsf{0} \cdot \mathsf{0}$ on cross-products (i.e., it is true of no cross-product), but obviously is not generally equal to $\mathsf{every} \cdot \mathsf{0} \cdot \mathsf{0}$, since we can construct non-cross-product relations on which it *is* true. However, by van Eijck's results we can also state a positive result, that it is in fact $(1, 2)$-reducible, equivalent to every·(*different*, *different*). Further, we know it cannot also be $(2, 1)$-reducible to some type (2) Q_1 and type (1) Q_2, or else by the diamond property there would exist some type (1) Q_3 making it 3-reducible to $\mathsf{every} \cdot \mathsf{Q}_3 \cdot \mathsf{Q}_2$, a contradiction.

6.2 The Frege Boundary and The Chomsky Hierarchy?

The above discussion on the characterization of the Frege boundary was initiated around the time the semantic automata were introduced. However, surprisingly these two perspectives have not been in much contact and there is still a major unanswered question: Where is the Frege boundary located in the Chomsky hierarchy? One could argue that irreducible languages are at least non-context-free assuming that for the language of a non-Fregean quantifier to even make sense the subwords (between $\boxdot$) must all have the same length. A simple pumping lemma argument demonstrates that no language with an arbitrary number of equal-length subwords is context-free. [17]

Looking at the problem from a somewhat different perspective, we saw a number of characterization results of the Frege boundary. So the question naturally arises: is the characterization of the Frege boundary *effective*? That is, given an arbitrary type (2) quantifier, can one effectively decide whether or not it is an iteration? The computational perspective allows us to ask this question as: given a language $L \subseteq \{0, 1, \boxdot\}$, is it decidable whether or not there are languages L_1 and L_2 such that $L = L_1 \cdot L_2$? (Recall Definition 3.3.) Steinert-Threlkeld [22] studies this problem and gives some partial answers, however, it seems like the conceptual challenge of proving a general result is still open. For instance, he shows that if $L \subseteq \{0, 1, \boxdot\}^*$ is a regular language then it is decidable whether there are regular languages L_1, L_2 in $\{0, 1\}$ such that $L = L_1 \cdot L_2$. He shows that the answer is positive in case of the regular quantifiers:

Theorem 6.8 ([22])**.** Let $L \subseteq \{0,1,\boxdot\}^*$ be a regular language. Then it is decidable whether there are regular languages L_1, L_2 in $\{0,1\}$ such that $L = L_1 \cdot L_2$.

The main obstacle to prove the decidability of iteration for context-free languages is that language equality is undecidable. This leads to the following conjecture: It is undecidable whether a given context-free language L in $\{0,1,\boxdot\}$ is an iteration of two context-free languages in $\{0,1\}$. However, as a corollary of Theorem 3.6 we have:

Corollary 6.9. It is decidable whether a given deterministic context-free language in $\{0,1,\boxdot\}$ is an iteration.

This discussion shows that the interaction between quantifiers and automata raises new and interesting questions in both domains (i.e., formal language theory and generalized quantifier theory). But there's a lot more to be done if we want to find a genuinely automata-theoretic characterization of the Frege boundary. Also, irreducible languages will come in different levels of difficulty. How can we further stratify languages of irreducible polyadic quantifiers in terms of the Chomsky hierarchy? Right now it seems that to make some progress on these issues one needs find suitable automata/language models and suitable representations (translation functions) [17, 22]. In other words, the following questions arise: Are there ways of representing models that are more appropriate to recognizing irreducible quantifiers? How would the languages of specific quantifiers be affected by such extensions, and how would the Frege boundary move up or down the Chomsky hierarchy as a result? And finally, we know hardly anything about the cognitive reality of the Frege boundary.

7 Conclusions

In the paper we have investigated semantic automata for polyadic quantifiers, extending recent results on quantifier iterations. First of all, we have given a new proof that regular and context-free languages are closed under quantifier iterations. Our proof emphasizes the explicit link between quantifier iteration and the standard concept of substitution known from formal language theory. Then we have proposed an explicit construction for minimal iteration DFA. Furthermore, we have provided a construction for the quantifier iteration of DCFL, hence, solving positively a natural question whether recently identified deterministic context-free quantifiers are closed under iteration.

Natural language also teems with *genuinely* polyadic quantification [13]. For example, none of the following are definable as or *reducible* to the iteration of any two monadic quantifiers, and thus are beyond the so-called Frege boundary [3, 14]:

1. *Three researchers together published five papers* (Cumulation[12])
2. *Not all twins are friends* (Resumption)
3. *Six hockey players punched each other* (Reciprocal)
4. *Every child read the same book* (Same)
5. *Most villagers and most townsmen hate each other* (Branching)

Given the recent extension to iteration, a next natural pursuit is to account for different forms of irreducible quantification with semantic automata. This leads to the question of where the Frege boundary lies in the Chomsky hierarchy. Some challenges to this pursuit are discussed in [17]. For example, the model translation described in this paper is just one possible function that is moreover particularly well-suited for encoding iterations, which are characterized by closure under *right-orientation*. Recall the model drawn in Figure 1: all that matters to the truth of an iteration is the *number* of outgoing arrows from each $a \in A$, and not *which* $b \in B$ the arrows point to. For irreducible polyadic quantification, the identities of the individuals in the model

[12]Cumulation is "on the boundary," definable as a Boolean combination of iterations: $(\mathsf{Q}_1 \cdot \mathsf{some})(A,B,R) \wedge (\mathsf{Q}_2 \cdot \mathsf{some})(B,A,R^{-1})$. The definition suggests the new translation function, more or less concatenating a string from $\tau_2(A,B,R)$ with a string from $\tau_2(B,A,R^{-1})$, and a formal automata construction and proof of REG and (D)CFL closure under cumulation is given in [17].

somehow contribute to the truth conditions. To identify the same element in different subwords, with the given translation, requires that all subwords have the same length (so we can identify elements by their position). But this alone makes irreducible quantifier languages far too complex, and certainly not context-free.

Future work will likely need to explore different model representations to move the project of identifying semantic automata for irreducible quantifiers forward. Additionally, it may be useful to explore automata models that have some sort of back-and-forth functionality, such as two-way automata, multihead and multitape automata, and automata with erasure, as a way of "having a finger on" more than one symbol in the input string at once.

The results in this paper and the possibility of future extensions also lead to many new empirical questions for cognitive modeling and formal learning theory. So far, experimentally measured difficulty in verification has neatly linked up with theoretical complexity, for both fine and more coarse-grained classifications of quantifiers.

We hope these contributions add momentum to the recent revival of interest in semantic automata, spurring further research into automata for polyadic quantifiers, and that the fruits of the algorithmic perspective on meaning may be brought to bear on practical applications related to multiquantifier constructions in natural language.

Acknowledgments

Jakub Szymanik was supported by the The Netherlands Organisation for Scientific Research Veni Grant NWO-639-021-232.

References

[1] Jon Barwise and Robin Cooper. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4(2):159–219, 1981.

[2] Johan van Benthem. *Essays in Logical Semantics*. D. Reidel Publishing Company, 1986.

[3] Johan van Benthem. Polyadic quantifiers. *Linguistics and Philosophy*, 12(4):437–464, 1989.

[4] Robin Clark. On the learnability of quantifiers. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 20, pages 911–923. Elsevier, 2^{nd} edition, 2011.

[5] Robin Clark and Murray Grossman. Number sense and quantifier interpretation. *Topoi*, 26(1):51–62, 2007.

[6] Paul Dekker. Meanwhile, within the Frege boundary. *Linguistics and Philosophy*, 26(5):547–556, 2003.

[7] Jan van Eijck. Normal forms for characteristic functions on n-ary relations. *Journal of Logic and Computation*, 15(2):85–98, 2005.

[8] Yuan Gao, Nelma Moreira, Rogério Reis, and Sheng Yu. A review on state complexity of individual operations. Technical report, Universidade do Porto, Technical Report Series DCC-2011-08, Version 1.1 (September 2012), `http://www.dcc.fc.up.pt/Pubs`, 2012.

[9] Nina Gierasimcauk. The problem of learning the semantics of quantifiers. *Lecture Notes in Computer Science*, 4363:117–126, 2007.

[10] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2^{nd} edition, 2001.

[11] Makoto Kanazawa. Monadic quantifiers recognized by deterministic pushdown automata. In *Proceedings of the 19^{th} Amsterdam Colloquium*, pages 139–146, 2013.

[12] E. Keenan. Further beyond the Frege boundary. In J. van der Does and J. van Eijck, editors, *Quantifiers, Logic, and Language*, CSLI Lecture Notes, pages 179–201. Stanford University, California, 1996.

[13] Edward L Keenan. Unreducible n-ary quantifiers in natural language. In Peter Gärdenfors, editor, *Generalized Quantifiers: Linguistic and Logical Approaches*, pages 109–150. D. Reidel Publishing Company, 1987.

[14] Edward L Keenan. Beyond the Frege boundary. *Linguistics and Philosophy*, 15(2):199–221, 1992.

[15] Donald E Knuth. On the translation of languages from left to right. *Information and Control*, 8(6):607–639, 1965.

[16] Per Lindström. First order predicate logic with generalized quantifiers. *Theoria*, 32(3):186–195, 1966.

[17] Sarah McWhirter. An automata-theoretic perspective on polyadic quantification in natural language. Master's thesis, University of Amsterdam, 2014. `http://www.illc.uva.nl/Research/Publications/Reports/MoL-2014-14.text.pdf`.

[18] Andrzej Mostowski. On a generalization of quantifiers. *Fundamenta Mathematicae*, 44:12–36, 1957.

[19] Marcin Mostowski. Computational semantics for monadic quantifiers. *Journal of Applied Non-Classical Logics*, 8(1-2):107–121, 1998.

[20] Stanely Peters and Dag Westerståhl. *Quantifiers in Language and Logic*. Clarendon Press, 2006.

[21] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2006.

[22] Shane Steinert-Threlkeld. On the decidability of iterated languages. In *Proceedings of Philosophy, Mathematics, Linguistics: Aspects of Interaction*, pages 215–224, 2014.

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

[24] Jakub Szymanik. Computational complexity of polyadic lifts of generalized quantifiers in natural language. *Linguistics and Philosophy*, 33(3):215–250, 2010.

[25] Jakub Szymanik and Marcin Zajenkowski. Comprehension of simple quantifiers: Empirical evaluation of a computational model. *Cognitive Science*, 34(3):521–532, 2010.

[26] Dag Westerståhl. Iterated quantifiers. In M. Kanazawa and Ch. Pinon, editors, *Dynamics, Polarity, and Quantification*, pages 173–209, 1994.