Iterating Semantic Automata

Shane Steinert-Threlkeld and Thomas F. Icard, III

April 26, 2013

Abstract

The semantic automata framework, developed originally in the 1980s, provides computational interpretations of generalized quantifiers. While recent experimental results have associated structural features of these automata with neuroanatomical demands in processing sentences with quantifiers, the theoretical framework has remained largely unexplored. In this paper, after presenting some classic results on semantic automata in a modern style, we present the first application of semantic automata to polyadic quantification, exhibiting automata for iterated quantifiers. We also discuss the role of semantic automata in linguistic theory and offer new empirical predictions for sentence processing with embedded quantifiers.[1]

1 Introduction

The interpretation of natural language determiner phrases as generalized quantifiers has led to deep and subtle insights into linguistic quantification. While the original goal of interpreting determiner phrases uniformly as higher-order properties is now seen as perhaps too simplistic,[2] the very idea that determiners can be assigned meanings which correctly predict their contribution to the meanings of sentences is of fundamental importance in semantics. Generalized quantifier theory, and arguably model-theoretic semantics in general, has largely developed independently of detailed questions about language processing. If one's aim is to understand how language can express truths, abstracting away from language users, then this orientation is arguably justified.[3] However,

[1]We thank Johan van Benthem, Christopher Potts, and Jakub Szymanik for helpful discussions and two anonymous referees for helpful comments.

[2]See Szabolcsi [2009] for an overview of some recent developments in quantifier theory. As she notes (p.5), "these days one reads more about what [generalized quantifiers] cannot do than about what they can."

[3]A classic statement of this approach to semantics can be found in Lewis [1970] (p.170): "I distinguish two topics: first, the description of possible languages or grammars as abstract semantic systems whereby symbols are associated with aspects of the world; and second, the description of the psychological and sociological facts whereby one of these abstract semantic systems is the one used by a person or population. Only confusion comes of mixing these two aspects." Lewis, Montague, and others clearly took the first as their object of study.

if the aim is to understand the role quantification plays in human cognition, model-theoretic interpretation by itself is too abstract. Patrick Suppes [1980] aptly summarized the point more than three decades ago:

> "It is a surprising and important fact that so much of language ... can be analyzed at a nearly satisfactory formal level by set-theoretical semantics, but the psychology of users is barely touched by this analysis." (p. 27)

Consider, for instance, the basic question of how a quantified sentence is verified as true or false. Generalized quantifier theory by itself has nothing to say about this question. One may worry that the psychological details would be too complex or unsystematic to admit useful and elegant theorizing of the sort familiar in formal semantics. However, in the particular case of verification, we believe the analysis of quantifier phrases by *semantic automata* provides a promising intermediate level of study between the abstract, ideal level of model theory and the mosaic, low-level details of processing.

Semantic automata, originally pioneered by Johan van Benthem [1986], offer an algorithmic, or procedural, perspective on the traditional meanings of quantifier phrases as studied in generalized quantifier theory. They are thus ideally suited to modeling verification-related tasks. A semantic automaton represents the *control structure* involved in assessing whether a quantified sentence is true or false. While there has been relatively little theoretical work in this area since van Benthem [1986] (though see Mostowski [1991, 1998]), a series of recent imaging and behavioral experiments has drawn on semantic automata to make concrete predictions about quantifier comprehension (McMillan et al. [2005, 2006], Szymanik and Zajenkowski [2010a,b]). These experiments establish (among other results, to be discussed further below) that working memory is recruited in the processing of sentences involving certain quantifiers, which corresponds to an analogous memory requirement on automata. Such studies provide impetus to revisit the semantic automata framework from a theoretical perspective. In this paper we extend the framework from simple single (monadic) quantifier sentences to sentences involving iterated quantification. This extension in turn raises new questions about processing.

In Section 2, we give a quick review of generalized quantifiers, followed by an extended introduction to semantic automata for single quantifier sentences in Section 3. Section 4 includes a more detailed discussion of how semantic automata might fit into semantic theorizing, at a level in between model-theoretic semantics and language processing. Finally, our main technical contribution is in Section 5 where we show how to extend the framework to iterations of quantifiers. A general construction method is given for combining automata for single quantifiers into automata for iterations. We then discuss further open empirical questions and other issues raised by this work.

2 Generalized Quantifiers

Definition 1 (Mostowski [1957], Lindström [1966])**.** A *generalized quantifier* Q of type $\langle n_1, \ldots, n_k \rangle$ is a class of models $\mathcal{M} = \langle M, R_1, \ldots, R_k \rangle$ closed under isomorphism, where each $R_i \subseteq M^{n_i}$.[4] A generalized quantifier is *monadic* if $n_i = 1$ for all i, and *polyadic* otherwise.

We write $Q_M\ R_1\ \ldots\ R_k$ as shorthand for $\langle M, R_1, \ldots, R_k \rangle \in Q$. Usually the subscripted M is omitted for readability. Thus, e.g., for type $\langle 1, 1 \rangle$ we write $Q\ A\ B$, where A and B are predicates. This connects with the more familiar definition given in linguistic semantics as can be seen by the following examples:

$$\begin{aligned} \textit{all} &= \{\langle M, A, B \rangle \mid A \subseteq B\} \\ \textit{some} &= \{\langle M, A, B \rangle \mid A \cap B \neq \varnothing\} \end{aligned}$$

The isomorphism closure condition (partially) captures the intuition that quantifiers are sensitive only to the size of the relevant subsets of M and not the identity of any particular elements or the order in which they are presented.

A useful classification of generalized quantifiers is given by the standard logical hierarchy. We restrict attention to type $\langle 1, 1 \rangle$, and we distinguish only between first-order and higher-order definability.

Definition 2. A generalized quantifier Q of type $\langle 1, 1 \rangle$ is *first-order definable* if and only if there is a first-order language $\mathcal{L}$ and an $\mathcal{L}$-sentence φ whose non-logical vocabulary contains only two unary predicate symbols A and B such that for any model $\mathcal{M} = \langle M, A, B \rangle$,

$$Q_M AB \quad \Leftrightarrow \quad \langle M, A, B \rangle \models \varphi.$$

The generalization to higher-order (non-first order) definability is obvious. As examples, *all*, *some*, and *at least three* are first-order definable:

$$\begin{aligned} \textit{all}_M AB &\Leftrightarrow \langle M, A, B \rangle \models \forall x\, (Ax \to Bx)\,; \\ \textit{some}_M AB &\Leftrightarrow \langle M, A, B \rangle \models \exists x\, (Ax \wedge Bx)\,; \\ \textit{at least three}_M AB &\Leftrightarrow \langle M, A, B \rangle \models \exists x, y, z\ \varphi(x, y, z), \end{aligned}$$

where $\varphi(x, y, z)$ is the formula

$$x \neq y \wedge y \neq z \wedge x \neq z \wedge Ax \wedge Bx \wedge Ay \wedge By \wedge Az \wedge Bz.$$

Most, *an even number of*, and *an odd number of* are (only) higher-order definable. For *most*, see, e.g., Appendix C of Barwise and Cooper [1981].

Because the space of type $\langle 1, 1 \rangle$ quantifiers places few constraints on possible determiner meanings, several properties have been offered as potential semantic universals, narrowing down the class of possible meanings. These properties

[4] For more complete introductions to the theory of generalized quantifiers, see Barwise and Cooper [1981], van Benthem [1986], Westerståhl [1989], Keenan [1996], Keenan and Westerståhl [2011].

seem to hold of (at least a majority of) quantifiers found in natural languages. Two of these will play a pivotal role in the development of semantic automata, due to their role in Theorem 1 below:

$$\begin{array}{rl} \textbf{CONS} & Q_M AB \text{ iff } Q_M A(A \cap B). \\ \textbf{EXT} & Q_M AB \text{ iff } Q_{M'} AB \text{ for every } M \subseteq M'. \end{array}$$

Lemma 1. *A quantifier Q satisfies* **CONS** + **EXT** *iff, for all $\mathcal{M} = \langle M, A, B\rangle$:*

$$Q_M AB \Leftrightarrow Q_A A(A \cap B).$$

Theorem 1. *A quantifier Q satisfies* **CONS** *and* **EXT** *if and only if for every M, M' and $A, B \subseteq M$, $A', B' \subseteq M'$, if $|A-B| = |A'-B'|$ and $|A\cap B| = |A'\cap B'|$, then $Q_M AB \Leftrightarrow Q_{M'} A'B'$.*

Proof. Suppose Q satisfies **CONS** and **EXT**. If $|A-B| = |A'-B'|$ and $|A\cap B| = |A' \cap B'|$, then we have bijections between the set differences and intersections which can be combined to give a bijection from A to A'. Thus $Q_A A(A \cap B) \Leftrightarrow Q_{A'} A'(A' \cap B')$ by isomorphism closure. By two applications of Lemma 1, $Q_M AB \Leftrightarrow Q_{M'} A'B'$.

In the other direction, for any given $\langle M, A, B\rangle$, let $M' = A' = A$ and $B' = A \cap B$. The assumption yields $Q_M AB \Leftrightarrow Q_{M'} A'B' \Leftrightarrow Q_A A(A \cap B)$, which by Lemma 1 implies **CONS** + **EXT**.

In other words, quantifiers that satisfy **CONS** and **EXT** can be summarized succinctly as binary relations on natural numbers. Given Q we define:

$$Q^c_M xy \quad \Leftrightarrow \quad \exists\, A, B \subseteq M \ \text{ s.t. } \ Q_M AB \text{ and } |A - B| = x, |A \cap B| = y.$$

Standard generalized quantifiers can thus be seen as particular simple cases.

$$\begin{aligned} \textit{every}^c_M\, xy &\Leftrightarrow x = 0 \\ \textit{some}^c_M\, xy &\Leftrightarrow y > 0 \\ \textit{at least three}^c_M\, xy &\Leftrightarrow y \geqslant 3 \\ \textit{most}^c_M\, xy &\Leftrightarrow y > x \\ \textit{an even number of}^c_M\, xy &\Leftrightarrow y = 2n \text{ for some } n \in \mathbb{N} \end{aligned}$$

Theorem 1 guarantees that the relation Q^c is always well defined.

2.1 Iterating Monadic Quantifiers

To handle sentences such as

(1) (a) One of our neighbors stole all but four of the sunflowers.

(b) Three explorers discovered most of the islands.

in which quantified phrases appear both in object position and subject position, we need to look at so-called polyadic lifts of monadic quantifiers. Intuitively, these sentences express complex properties of the respective transitive verbs. Since these verbs take two arguments, it will be impossible to give truth-conditions using monadic predicates.

In particular, we will need iterations of type $\langle 1, 1\rangle$ quantifiers. For notation, if R is a binary relation, we write

$$R_x = \{y \mid Rxy\}$$

If Q_1 and Q_2 are type $\langle 1, 1\rangle$, then $It(Q_1, Q_2)$ will be of type $\langle 1, 1, 2\rangle$, defined:

$$It(Q_1, Q_2)\ A\ B\ R \quad \Leftrightarrow \quad Q_1\ A\ \{x \mid Q_2\ B\ R_x\}$$

We will sometimes use the alternative notation $Q_1 \cdot Q_2$ for $It(Q_1, Q_2)$.[5]

Sentences with embedded quantifiers can be formalized as iterations. For instance, the sentences in (1) would typically be assigned truth conditions in (2):

(2) (a) $It(\textit{some}, \textit{all\_but\_four})$ *neighbor sunflowers stole*

(b) $It(\textit{three}, \textit{most})$ *explorers islands discovered*

For example, (2) (a) holds iff

$$\textit{neighbor} \cap \{x \mid \textit{all\_but\_four sunflowers stole}_x\} \neq \varnothing$$

In Section 5 we will show how to define automata corresponding to such iterations. But first, in the next section, we introduce automata for single quantifier sentences.

3 Semantic Automata

Throughout this section all quantifiers are assumed to satisfy **CONS** and **EXT**.[6] The basic idea behind semantic automata is as follows: given a model $\mathcal{M} = \langle M, A, B\rangle$ and an enumeration of A, we define a string in $s \in \{0, 1\}^*$ by assigning 0 to elements of $A \backslash B$ and 1 to $A \cap B$. Note that we can use any enumeration of A since quantifiers are closed under isomorphism. To ensure that these strings are finite, we consider only finite models. It then follows by Theorem 1 that

$$\mathcal{M} \in Q \quad \Leftrightarrow \quad \langle \#_0(s), \#_1(s)\rangle \in Q^c,$$

[5]This is a special case of a general definition for iterating quantifiers. For details, see Chapter 10 of Peters and Westerståhl [2006].

[6]These assumptions can be dropped. But with them, we can use a two-letter alphabet in defining our languages below. Without them, we would need a four-letter alphabet. Moreover, little to no coverage of natural language determiners is lost by making these assumptions.

where $\#_0$ and $\#_1$ are recursively defined functions yielding the number of zeros and of ones in a string, respectively. The goal is to define machines that correspond to quantifiers, in the sense that they accept exactly the strings encoding models in the denotation of the quantifier. The *language of* Q is the set

$$\mathcal{L}_Q = \left\{ s \in \{0,1\}^* \mid \langle \#_0(s), \#_1(s) \rangle \in Q^c \right\}.$$

Definition 3. Let $\mathcal{M} = \langle M, A, B \rangle$ be a model, $\vec{a}$ an enumeration of A, and $n = |A|$. We define $\tau(\vec{a}, B) \in \{0,1\}^n$ by

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

Thus, τ defines the string corresponding to a particular finite model.

Lemma 2. *If a quantifier Q satisfies* **CONS** *and* **EXT**, *then the language $\mathcal{L}_Q$ is permutation-invariant. In other words, if $\langle M, A, B \rangle \in Q$, then $\tau(\vec{a}, B) \in \mathcal{L}_Q$ for all enumerations $\vec{a}$ of A.*

Proof. Membership in $\mathcal{L}_Q$ depends only on the number of ones and zeros in a string, neither of which is affected by permutations.

The simplest class of automata are the (deterministic) finite-state automata:[7]

Definition 4. A *deterministic finite-state automaton (DFA)* is a tuple $\langle Q, \Sigma, \delta, q_0, F \rangle$:

- Q a finite set of *states*
- $\delta : Q \times \Sigma \to \Sigma$ a *transition function*
- $F \subseteq Q$ the set of *accepting states*
- Σ a finite set of *input symbols*
- q_0 the *start state*

We denote the components of a DFA M by $Q(\mathsf{M})$, $\Sigma(\mathsf{M})$, etc.

A DFA can be given a simple graphical representation. Each state $q \in Q$ corresponds to a node. We will often represent these as circles, omitting the name of the node. Final states (another name for accepting states) in F will be represented as double circles. If $\delta(q, a) = p$, we draw a directed arc from q to p labeled by a. For semantic automata our alphabet Σ will always be $\{0, 1\}$.

For an example, consider *every*. Recall that $every^c_M\, xy \Leftrightarrow x = 0$. Thus, the only words $w \in \{0,1\}^*$ that should be accepted are those where $\#_0(w) = 0$. So the automaton for *every* will start in an accepting state and stay there so long as it only reads 1s. As soon as it reads a 0, however, the automaton moves to a non-accepting state and remains there. We represent *every* by the DFA in Figure 1.

[7]For a canonical reference on automata theory, see Hopcroft et al. [2001].

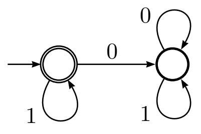


Figure 1: A finite state automaton for *every*.

Here is a toy example: $A = \{a, b\}$, $B = \{a, b, c\}$. In this case, A will be represented by the string 11 and the DFA for *every* will start and stay in the accepting state. If, on the other hand, $B = \{a, c\}$, then A will be represented by the string 10. Upon reading 0, the DFA for *every* will move to the non-accepting state and end there.

Recall that $some^c_M\, xy \Leftrightarrow y > 0$. Therefore, a DFA for *some* should accept any word $w \in \{0,1\}^*$ which contains at least one 1 (i.e. such that $\#_1(w) > 0$). It is easily verified that the DFA depicted in Figure 2 will do just that.

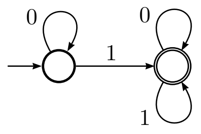


Figure 2: A finite state automaton for *some*.

One can prove that all first-order definable quantifiers have languages that can be accepted by DFAs. Motivated by this result, one might hope that only first-order definable quantifiers can be so simulated. It turns out, however, that some higher-order definable quantifiers can also be modeled by finite state automata. Figure 3 shows such an automaton for *an even number of*, which is not first-order definable.

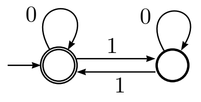


Figure 3: A cyclic finite state automaton for *an even number of*.

Switching the end-state from the node on the left to the node on the right renders Figure 3 a (cyclic) finite state automaton for *an odd number of*. In general, the first-order definable quantifiers correspond to a smaller class of DFAs:

Theorem 2 (van Benthem [1986], p.156-157)**.** *A quantifier Q is first-order definable iff $\mathcal{L}_Q$ can be recognized by a permutation-invariant acyclic finite state automaton.*

Moreover, it is not the case that all higher-order quantifiers can be simulated by cyclic finite-state automata. Mostowski [1991] (see also Mostowski [1998]) has characterized the class of quantifiers which can. The type $\langle 1 \rangle$ *divisibility quantifier* D_n is defined:

$$\langle M, A \rangle \in D_n \quad \text{iff} \quad |A| \text{ is divisible by } n.$$

Theorem 3 (Mostowski [1991])**.** *Finite state automata accept exactly the class of quantifiers of type* $\langle 1, \ldots, 1 \rangle$ *definable in first-order logic augmented with* D_n *for all* n*.*

To simulate quantifiers such as *less than half* or *most*, neither of which is definable in divisibility logic, we must move to the next level of the machine hierarchy, to pushdown automata.[8] Intuitively, a pushdown automaton augments a DFA with a stack of memory; this stack is a last-in/first-out data structure, onto which we can "push" new content, and from which we can "pop" the topmost element.

Definition 5. A (non-deterministic) *pushdown automaton (PDA)* is a tuple $\langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$:

- Q is a finite set of *states*
- Γ is a finite *stack alphabet*
- q_0 is the *start state*
- Z_0 is the *start symbol*
- Σ is a finite set of *input symbols*
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$ is a *transition function*
- F is the set of *accepting states*

The biggest difference between DFAs and PDAs lies in the transition function. The idea is that δ receives as input the current state, the symbol most recently read, and the symbol at the top of the stack. An output pair (p, γ) indicates that the automaton has moved to state p and replaced the top of the stack with the string γ. Suppose X is the symbol at the top of the stack. If $\gamma = \epsilon$ (here ϵ denotes the empty string), then the top of the stack has been popped. If $\gamma = X$, then no change has been made. If $\gamma = YZ$, then X has been replaced with Z and Y has been pushed onto the stack. While the definition of a PDA allows for any length string to be pushed, we will incidentally work only with strings of length 2.

[8] In particular, we are moving one level up the Chomsky [1959] hierarchy of formal grammars. The *regular* languages are generated by DFAs, while the *context-free* languages are generated by pushdown automata. That quantifiers such as *most* and *less than half* are not computable by DFAs can be easily proven using the Pumping Lemma for regular languages.

Graphically, we represent $\delta(q, a, X) = (p, \gamma)$ by a directed arc from q to p labeled by $a, X/\gamma$. Here X/γ is intended to signify that symbol X has been replaced by string γ at the top of the stack. We use x as a variable over characters in an alphabet in order to consolidate multiple labels. In all of the following examples, we assume $\Gamma = \Sigma \cup \{\epsilon\}$.

Figure 4 depicts a PDA for *less than half.* The idea here is that we push 1s

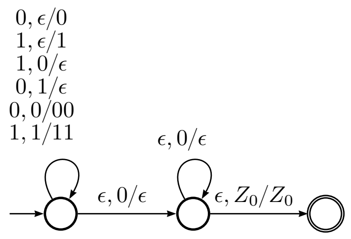


Figure 4: A pushdown automaton for *less than half.*

and 0s to the stack as often as we can, but popping off pairs. If a 1 is read and a 0 is on the stack, we pop the 0 and *vice versa.* This has the affect of "pairing" the members of $A \backslash B$ and $A \cap B$. Because *less than half* holds when the former outnumber the latter, there should be only 0s on the stack at the end of this process. Therefore, the transition to the accepting state occurs only when the string of 0s and 1s has been entirely processed and popping off any remaining 0s exhausts the contents of the stack. Modifying the labels on the edges of the final two states to pop all 1s would render this a PDA for *most.*

The final result reported in this section depends on one final definition.

Definition 6. A quantifier Q is *first-order additively definable* if there is a formula φ in the first-order language with equality and an addition symbol $\bar{+}$ such that

$$Q_M^c ab \quad \Leftrightarrow \quad \langle \mathbb{N}, +, a, b \rangle \models \varphi(a, b)$$

Theorem 4 (van Benthem [1986], p.163-165)**.** *$\mathcal{L}_Q$ is computable by a pushdown automaton if and only if Q is first-order additively definable.*

4 Automata and Processing

How exactly does this work on automata relate to questions about processing? On one hand, the machine representations of quantifiers discussed in the previous section are inspired directly by the standard model-theoretic meanings assumed in classical quantifier theory. On the other hand, the fine structure of these representations promises a potential bridge to lower level questions about language processing. In this section we discuss two important dimensions of this connection:

(1) Semantic automata suggest a relatively clean theoretical separation between semantic competence and performance errors.

(2) Recent imaging and behavioral experiments have revealed that structural differences in automata may be predictive of neuroanatomical demands in quantifier processing.

4.1 Explaining Performance Errors

The logical theory of generalized quantifiers is occasionally dismissed as being irrelevant to the psychological facts about how people produce and assess quantified sentences, typically by citing data that suggests human reasoning with quantifiers does not match the patterns assumed in standard logical analysis. Indeed, experiments reveal people diverge from standard logical prescriptions, not only in reasoning tasks such as the classical syllogism (see, e.g. Chater and Oaksford [1999]), but even in simple verification tasks (see, e.g. McMillan et al. [2005] with a similar pattern in Szymanik and Zajenkowski [2010a]). One could take this to show, or at least to reinforce the idea, that logical/truth-conditional semantics and the psychology of language are best kept separate, with the former studying abstract normative aspects of meaning and the latter studying the actual mechanisms involved in processing.[9] Yet the normative aspects of meaning, and of quantifier meanings in particular, are clearly relevant to questions about how people use quantifiers, and *vice versa*. While a complete discussion of this issue is beyond the scope of this paper, we would like to point out that semantic automata have the potential to serve as a natural bridge. In principle, they allow for a separation between abstract *control structure* involved in quantifier verification and innumerable other variables that the framework leaves underspecified: order of evaluation, predication judgments, domain restriction, and any contextual or practical factors that might affect these and other variables. This could be viewed as a modest distinction between *competence* and *performance* for quantifier expressions.[10]

We might hypothesize that competence with a particular quantifier involves, at least in part, internalizing the right abstract computational mechanism for verification, in particular that given by the appropriate automaton. How precisely verification is implemented on a given occasion will depend on many factors quite independent from the meanings of quantifiers: prototype effects, saliency effects, time constraints, and so on. Consider, for instance, how one might verify or falsify a sentence like (1):

All U.S. presidents have served at least one year in office. (1)

[9]Recall the quotation from Lewis [1970] in Footnote 3 above.

[10]Our suggestion is compatible with many interpretations of what this distinction comes to. For instance, the relatively non-committal interpretation of Smolensky [1988] says competence of a system or agent is "described by hard constraints" which are violable and hold only in the ideal limit, with unbounded time, resources, and other enabling conditions. The actual implementational details are to be given by "soft constraints" at a lower level, which have their own characteristic effects on performance.

Supposing one does not already know whether this is true or false, but that an answer must be computed using information stored in memory, one might check famous presidents like George Washington or Abraham Lincoln first and only then move to less salient presidents. Alternatively, if one actually knew James Garfield or William Harrison had served less than a year, such information might be retrieved quickly without first checking more salient presidents. The subtleties of such strategies are fascinating, but arguably go beyond the meanings of quantifiers. Indeed, they arise in tasks and phenomena having nothing to do with quantifiers.

The same can be said for cases where a search is terminated too soon. In example (1) a person might think about a few salient examples and conclude that the sentence is true. For a particular task it may take too long to think about all forty-four presidents, or it may not be worth the effort, and a quick guess is sufficient. However, even in such cases, upon being shown a counterexample, a subject will not insist that the sentence is actually true just because they were unable to identify the counterexample. It is in that way that people are reasonably attuned to the proper, "normative" meanings. Ordinary speakers have a good sense for what needs to be checked to verify a quantified sentence, even if in practice going through the necessary steps is difficult or infeasible. Semantic automata allow separating out control structure from the specific algorithm used to implement the procedure.

In terms of the Marr's famous *levels of explanation* (Marr [1982]), the standard model-theoretic semantics of quantification could be seen as a potential computational, or level 1, theory. Semantic automata offer more detail about processing, but, as we have just argued, less than one would expect by a full algorithmic story about processing (level 2), which would include details about order, time, salience, etc. Thus, we might see the semantic automata framework as aimed at level 1.5 explanation,[11] in between levels 1 and 2, providing a potential bridge between abstract model theory and concrete processing details.

4.2 Experimental Results

While this separation between competence and performance remains somewhat speculative, recent experimental work has shown that certain structural features of semantic automata are concretely reflected in the neuroanatomical demands on quantifier verification. Recall that certain quantifiers can be computed by memoryless finite state automata whereas others (such as *most*) require a pushdown automaton which has a form of memory in its stack data structure. McMillan et al. [2005] used fMRI to test "the hypothesis that all quantifiers recruit inferior parietal cortex associated with numerosity, while only higher-

[11] Peacocke [1986] coined the term "level 1.5", though in a slightly different context. Soames [1984] independently identified three levels of (psycho)linguistic investigation which appear to (inversely) correlate with Marr's three levels. Pietroski et al. [2009] also articulate the idea that verification procedures for quantifiers (for *most*, specifically) provide a level 1.5 explanation. Independently, the idea of level 1.5 explanation has recently gained currency in Bayesian psychology in relation to *rational process models* (personal communication, Noah Goodman).

order quantifiers recruit prefrontal cortex associated with executive resources like working memory." In their study, twelve native English speakers were presented with 120 grammatically simple sentences using a quantifier to ask about a color feature of a visual array. The 120 sentences included 20 instances of 6 different quantifiers: three first-order (*at least 3*, *all*, *some*) and three higher-order (*less than half*, *an odd number of*, *an even number of*). Each subject was first shown the proposition alone on a screen for 10s, then the proposition with a visual scene for 2500ms, then a blank screen for 7500ms during which they were to assess whether the proposition accurately portrayed the visual scene.

While behavioral results showed a statistically significant difference in accuracy between verifying sentences with first-order quantifiers and those with higher-order quantifiers, more interesting for our present purposes was the fact that activation in brain regions (dorsolateral prefrontal and inferior frontal cortices) typically linked with executive functioning such as working memory was found only in processing higher-order quantifiers. They concluded that the formal difference between the machines required to compute quantifier languages seems to reflect a physical difference in neuro-anatomical demands during quantifier comprehension.

This first piece of evidence does not tell the whole story, however. The first-order vs. higher-order distinction does not map directly on to the distinction between DFAs and PDAs because of parity quantifiers such as *an even number of*, which are computable by cyclic DFAs. Szymanik [2007] observed that McMillan et al. did not make this distinction and began investigating the demands placed on memory by parity quantifiers. In a subsequent study by Szymanik and Zajenkowski [2010b] reaction times were found to be lowest for Aristotelian quantifiers (*all* , *some*), higher for parity (*an even number of*), yet higher for cardinals of high rank (*at least 8*), and highest for proportionality quantifiers (*most*). This provides some evidence that the complexity of the minimal automaton, and not simply the kind of automaton, may be relevant to questions about processing.[12] A subsequent study by Szymanik and Zajenkowski [2011] showed that proportionality quantifiers place stronger demands on working memory than parity quantifiers. This result is consistent with the semantic automata picture, since a PDA computing the relevant parity quantifiers never needs to push more than one symbol on to the stack.

Finally, it is also relevant that McMillan et al. [2005] found no significant difference in activation between judgments involving *at least 3* where the relevant class had cardinality near or distant to three. This suggests subject are invoking a precise number sense in such cases. This contrasts with recent studies by Pietroski et al. [2009] and Lidz et al. [2011], which attempt to distinguish which of several verification procedures are actually used when processing sentences with *most*. Although the present paper shares much in spirit with this project, their protocols show a scene for only 150 or 200ms, which effectively forces the use of the *approximate number system*.[13] The finding is that in such

[12]Indeed, a general notion of complexity for automata in the context of language processing would be useful in this context (see also the discussion in Section 5.1.1).

[13]See Dehaene [1997] for the distinction between precise and approximate number systems.

cases people do not seem to be using a pair-matching procedure such as that defined above. We conjecture that with more time the pair-matching algorithm would be used, at least approximately.

The experimental work described in this section has been based solely on automata defined for monadic quantifiers. In order to carry this project further, and to understand its potential and its shortcomings as a high-level processing model, we need to define machines appropriate for more complex quantificational types. In the next section we take an important first step in this direction, defining automata for iterations of quantifiers.[14]

5 Iteration

We now show how to define automata for type $\langle 1, 1, 2\rangle$ iterations of quantifiers by composing the automata already defined for $\langle 1, 1\rangle$ quantifiers. Recall:

$$It(Q_1, Q_2)\ A\ B\ R \quad \Leftrightarrow \quad Q_1\ A\ \{x \mid Q_2\ B\ R_x\}$$

Intuitively, we simply want to run the automaton for Q_1 on the string generated by the sets A and $\{x \mid Q_2\ B\ R_x\}$. The trick, however, comes in "generating" this second set on the fly. Our basic maneuver will be this: we run the Q_2 automaton on B and R_{a_i} for every $a_i \in A$. For each run, we push onto a stack a 1 or a 0 corresponding to whether $a_i \in \{x \mid Q_2\ B\ R_x\}$ or not. Then, we run a transformed version of the Q_1 automaton where every transition has been replaced with one that pops symbols off the stack instead of reading them. In this sense, the stack of the iterated machine will serve as the input tape for the Q_1 machine and we will use the Q_2 machine to generate an appropriate string. We will now make this all precise, first working with quantifiers computable by finite state automata, and then generalizing to those computable only by pushdown automata.

Definition 7. Let $\mathcal{M} = \langle M, A, B, R\rangle$ be a model, $\vec{a}$ and $\vec{b}$ enumerations of A and B, with $n = |A|,\ m = |B|$. We overload notation (see Definition 3) by allowing τ to take a relation as an extra argument:

$$\tau\left(\vec{a}, \vec{b}, R\right) = \left(\tau\left(\vec{b}, R_{a_i}\right) \boxdot\right)_{i \leqslant n}$$

where $\tau\left(\vec{b}, R_{a_i}\right)$ is the translation given in Definition 3. The operation $(\cdot)_{i\leqslant n}$ concatenates instances of $(\cdot)$ for $0, \ldots, n$. The $\boxdot$ functions as a separator symbol in a way that will shortly be made precise.

To see this translation in a concrete example, consider a model $\langle M, A, B, R\rangle$ where $M = \{x, y, z\}$, $A = \{x, y\}$, $B = M$, and

$$R = \{\langle x, y\rangle, \langle y, x\rangle, \langle y, y\rangle, \langle y, z\rangle\}$$

[14]Szymanik [2010] investigated the computational complexity of polyadic lifts of monadic quantifiers. This approach, however, deals only with Turing machines. Our development can be seen as investigating the fine structure of machines for computing quantifier meanings.

Let the enumerations $\vec{a}$ and $\vec{b}$ be given alphabetically. Then $\tau\left(\vec{a},\vec{b},R\right)$ will be

$$010 \boxdot 111\boxdot \tag{2}$$

Definition 8. Let Q_1 and Q_2 be quantifiers of type $\langle 1,1\rangle$. We define *the language of* $Q_1 \cdot Q_2$ by

$$\begin{aligned}\mathcal{L}_{Q_1\cdot Q_2} = \ &\{w \in (w_i\boxdot)^* \mid i \leqslant n,\ w_i \in \{0,1\}^* \text{ and} \\ &\left\langle \operatorname{card}\left(\{w_i \mid w_i \notin \mathcal{L}_{Q_2}\}\right), \operatorname{card}\left(\{w_i \mid w_i \in \mathcal{L}_{Q_2}\}\right)\right\rangle \in Q_1^c\}\end{aligned}$$

For $w \in (w_i\boxdot)^*$, we write *numsep* (w) for the number of $\boxdot$ symbols in w.

Note that $w \notin \mathcal{L}_Q$ iff $w \in \mathcal{L}_{\neg Q}$ where $\neg Q$ is the outer negation, given by

$$\langle M, A, B\rangle \in \neg Q \Leftrightarrow \langle M, A, B\rangle \notin Q.$$

We illustrate the definition with a few examples.

Example 1. Here are a few examples using *some*, *every*, and *most*. We omit some of the conditions (e.g., that $i \leqslant n$) to enhance readability.

$$\begin{aligned}
w \in \mathcal{L}_{every\cdot some} &\Leftrightarrow \left\langle \operatorname{card}\left(\{w_i \mid w_i \in \mathcal{L}_{some}\}\right), \operatorname{card}\left(\{w_i \mid w_i \notin \mathcal{L}_{some}\}\right)\right\rangle \in every^c \\
&\Leftrightarrow \operatorname{card}\left(\{w_i \mid w_i \notin \mathcal{L}_{some}\}\right) = 0 \\
&\Leftrightarrow \operatorname{card}\left(\{w_i \mid \#(1) = 0\}\right) = 0 \\
w \in \mathcal{L}_{some\cdot every} &\Leftrightarrow \operatorname{card}\left(\{w_i \mid \#(0) = 0\}\right) > 0 \\
w \in \mathcal{L}_{most\cdot some} &\Leftrightarrow \operatorname{card}\left(\{w_i \mid \#(1) > 0\}\right) > \operatorname{card}\left(\{w_i \mid \#(1) = 0\}\right)
\end{aligned}$$

One can see that the translation given in (2) will be in $\mathcal{L}_{some\cdot every}$.

As before we have the following relationship between translations of models and languages.

Proposition 1. *Let $\mathcal{M} = \langle M, A, B, R\rangle$ be a model and Q_1, Q_2 quantifiers of type $\langle 1,1\rangle$. Then for any enumerations $\vec{a}$ and $\vec{b}$ of A and B,*

$$\tau\left(\vec{a},\vec{b},R\right) \in \mathcal{L}_{Q_1\cdot Q_2} \Leftrightarrow \langle M, A, B, R\rangle \in It(Q_1, Q_2)$$

5.1 Iterating Finite State Machines

In order to define PDAs that accept these iterated languages, we first need to define the aforementioned transformation on DFAs which allows a stack to be treated as if it were input.

Definition 9. Let M be a DFA. The *pushdown reader of* M, M^p is defined by

- $Q\left(\mathsf{M}^p\right) = Q\left(\mathsf{M}\right)$, $q_0\left(\mathsf{M}^p\right) = q_0$, $F\left(\mathsf{M}^p\right) = F\left(\mathsf{M}\right)$;
- $\Sigma\left(\mathsf{M}^p\right) = \varnothing$;

- $\Gamma(\mathsf{M}^p) = \Sigma(\mathsf{M})$;
- $\delta(\mathsf{M}^p) = \{\langle q_1, \epsilon, r, \epsilon, q_2\rangle \mid \langle q_1, r, q_2\rangle \in \delta(\mathsf{M})\}$.

In other words, the stack alphabet of the pushdown reader is the input alphabet of the original automaton. The state spaces are the same, but each transition $q_1 \xrightarrow{r} q_2$ in M is replaced by $q_1 \xrightarrow{\epsilon, r/\epsilon} q_2$, i.e. by popping an r from the stack. On its own, a pushdown reader is a fairly meaningless machine since its input alphabet is empty. But they will prove to be a critical component in the PDAs which compute iterated quantification.

Before defining the iteration automata, we provide several helper definitions. For an automaton M, let the *sign of* $q \in Q(\mathsf{M})$ be given by

$$sgn(q) = \begin{cases} 1 & q \in F \\ 0 & q \notin F \end{cases}$$

We define the *sign of* M as

$$sgn(\mathsf{M}) = sgn(q_0(\mathsf{M}))$$

In what follows, the complement operator $(\cdot)^c : \{0,1\} \to \{0,1\}$ maps 1 to 0 and 0 to 1. With these definitions in hand, we can proceed to the central definition of this paper.

Definition 10 (Iteration Automaton)**.** Let Q_1 and Q_2 be two DFAs accepting $\mathcal{L}_{Q_1}$ and $\mathcal{L}_{Q_2}$, respectively. The PDA $\mathsf{It}(\mathsf{Q}_1, \mathsf{Q}_2)$ is given by:

- $Q = \{q_I\} \cup Q(\mathsf{Q}_1^p) \cup Q(\mathsf{Q}_2)$
- $\Sigma = \{0, 1, \boxdot\}$
- $\Gamma = \{0, 1\}$
- Transition function:

$$\begin{aligned}
\delta = \;& \delta(\mathsf{Q}_1^p) \\
& \cup \{\langle q_I, \varepsilon, x, sgn(\mathsf{Q}_2)\,x, q_0(\mathsf{Q}_2)\rangle \mid i \leqslant n\} \\
& \cup \{\langle q_1, 1, x, x, q_2\rangle \mid \langle q_1, 1, q_2\rangle \in \delta(\mathsf{Q}_2) \text{ and } sgn(q_1) = sgn(q_2)\} \\
& \cup \{\langle q_1, 0, x, x, q_2\rangle \mid \langle q_1, 0, q_2\rangle \in \delta(\mathsf{Q}_2) \text{ and } sgn(q_1) = sgn(q_2)\} \\
& \cup \{\langle q_1, 1, x, x^c, q_2\rangle \mid \langle q_1, 1, q_2\rangle \in \delta(\mathsf{Q}_2) \text{ and } sgn(q_1) \neq sgn(q_2)\} \\
& \cup \{\langle q_1, 0, x, x^c, q_2\rangle \mid \langle q_1, 0, q_2\rangle \in \delta(\mathsf{Q}_2) \text{ and } sgn(q_1) \neq sgn(q_2)\} \\
& \cup \{\langle q, \boxdot, x, x, q_I\rangle \mid q \in Q(\mathsf{Q}_2)\} \\
& \cup \{\langle q_I, \varepsilon, x, x, q_0(\mathsf{Q}_1^p)\rangle\}
\end{aligned}$$

- $q_0 = q_I$
- $F = F(\mathsf{Q}_1^p)$

The basic idea is as follows: q_I is a new start state. From q_I, we have a ε transition to the start state of Q_2. When we take such a transition, a 1 or a 0 is pushed onto the stack according to whether or not the start state of Q_2 is an accepting state. The role of *sgn* and $(\cdot)^c$ is to ensure that we switch the original symbol pushed onto the stack by the i transition whenever we go from an accepting to a non-accepting state or *vice versa*. In this way, we push exactly one symbol on to the stack for each visit to Q_2: a 1 if it ended in an accepting state and a 0 if not. The $\boxdot$ transitions from each state of Q_2 to q_I enable $\boxdot$ to function as a separating symbol. From q_I, we can also take an ε-transition to Q_1^p; this pushdown reader will then process the stack generated by the visits to Q_2.

Example 2. In Figure 5 is a PDA for computing *some* $\cdot$ *every*.

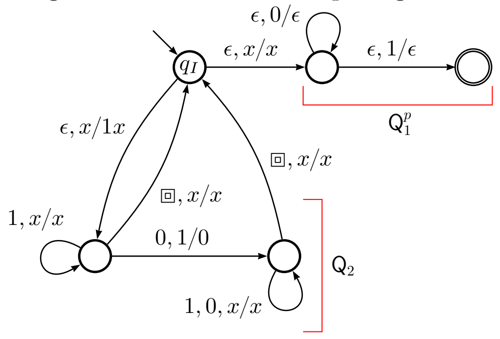


Figure 5: A pushdown automaton for *some A R every B*.

Here, Q_1^p is the pushdown reader (see Definition 9) of some. Q_2 is the transformed copy of every. Note that we push a 1 onto the stack on the transition from q_I to the start state of every since $q_0(\mathsf{every}) \in F(\mathsf{every})$. Similarly, we pop this 1 and push a 0 on the 0 transition since this goes from an accepting to rejecting state of every.

Consider our earlier string

$$010 \boxdot 111 \boxdot$$

which we know to be in $\mathcal{L}_{some \cdot every}$. When reading 010, this automaton will push a 0 on to the stack, but will push a 1 on to the stack when it reads 111. Thus, some^p will accept the stack input and so the whole string will be accepted.

We record here a basic fact about iterated machines which follows straightforwardly from the definition, and which will be important shortly.

Fact 1. $\mathsf{It}(\mathsf{Q}_1, \mathsf{Q}_2)$ has $1 + |Q(\mathsf{Q}_1)| + |Q(\mathsf{Q}_2)|$ states.

While the informal description of $\mathsf{It}(\mathsf{Q}_1, \mathsf{Q}_2)$ and the example make it seem plausible that this PDA accepts the right iterated language, we now make this equivalence precise. First, we prepare a few preliminary results, for which a basic definition of the notion of computation in a PDA is required.

Definition 11. Given a PDA M, a triple $\langle q, w, X\rangle \in Q \times \Sigma^* \times \Gamma^*$ is called an *instantaneous description* of M, specifying the current state, what of the input has not been read, and the current stack contents. The transition function defines a notion of one-step computation: for every $\langle q_1, x, A, q_2, X\rangle \in \delta$, we write

$$\langle q_1, xw, AY\rangle \vdash_{\mathsf{M}} \langle q_2, w, XY\rangle$$

for every $w \in \Sigma^*$ and $Y \in \Gamma^*$, with $\vdash_{\mathsf{M}}^*$ the reflexive, transitive closure of $\vdash_{\mathsf{M}}$.

Intuitively, $\langle q_1, w_1 w_2, AY\rangle \vdash_{\mathsf{M}}^* \langle q_2, w_2, XY\rangle$ means that there is a sequence of transitions starting in q_1 which reads w_1, ends in q_2 and changes the stack from AY to XY.

Lemma 3. *Let Q_1 and Q_2 be quantifiers corresponding to regular languages and $w_i \in \{0,1\}^*$. Abbreviate $\mathsf{It}(\mathsf{Q}_1, \mathsf{Q}_2)$ by M.*

(1) If $w_i \in \mathcal{L}_{Q_2}$, then $(q_I, w_i \boxdot w, X) \vdash_{\mathsf{M}}^* (q_I, w, 1X)$ for any $X \in \Gamma^*$, $w \in \Sigma^*$.

(2) If $w_i \notin \mathcal{L}_{Q_2}$, then $(q_I, w_i \boxdot w, X) \vdash_{\mathsf{M}}^* (q_I, w, 0X)$ for any $X \in \Gamma^*$, $w \in \Sigma^*$.

In other words, for any string w_i, there is a $w_i\boxdot$ path through the iterated PDA such that a 1 or 0 is pushed onto the stack according to whether or not $w_i \in \mathcal{L}_{Q_2}$.

Proof. We prove (1) by induction on the length of w_i. The proof for (2) is wholly analogous. Assume $w_i \in \mathcal{L}_{Q_2}$.

If $|w_i| = 0$ (i.e. $w_i = \epsilon$), then $q_0(\mathsf{Q}_2) \in F(\mathsf{Q}_2)$, i.e. $sgn(\mathsf{Q}_2) = 1$. Thus, we take the $\epsilon, X/1X$ transition from q_I to $q_0(\mathsf{Q}_2)$, immediately followed by the $\boxdot, x/x$ transition back to q_I.

For the inductive step, let $|w_i| = n$ and write $w_i = w_i^- c_i$ where $c_i \in \{0,1\}$. By assumption, Q_2 accepts w_i, so its w_i sequence of transitions ends in some $q_{w_i} \in F(\mathsf{Q}_2)$. We need to check two cases: $w_i^- \in \mathcal{L}_{Q_2}$ or $w_i^- \notin \mathcal{L}_{Q_2}$.

If $w_i^- \in \mathcal{L}_{Q_2}$, then by the inductive hypothesis, there is a $w_i^-\boxdot$ sequence in the iterated PDA sending X to $1X$. Moreover, by assumption, $sgn\left(q_{w_i^-}\right) = sgn(q_{w_i})$. Thus, we replace the $\boxdot$ transition in the $w_i^-\boxdot$ sequence with the $c_i, x/x$ transition from $q_{w_i^-}$ to q_{w_i} that is given by definition of $\delta(\mathsf{It}(\mathsf{Q}_1, \mathsf{Q}_2))$.

If $w_i^- \notin \mathcal{L}_{Q_2}$, then by the inductive hypothesis, there is a $w_i^-\boxdot$ sequence in the iterated PDA sending X to $0X$. This time, by assumption, $sgn\left(q_{w_i^-}\right) \neq sgn(q_{w_i})$. Thus, we replace the $\boxdot$ transition in the $w_i^-\boxdot$ sequence with the $c_i, 0/1$ transition from $q_{w_i^-}$ to q_{w_i} that is given by definition of $\delta(\mathsf{It}(\mathsf{Q}_1, \mathsf{Q}_2))$.

Corollary 1. Let $w \in (w_i\boxdot)^*$ where $w_i \in \{0,1\}^*$. Then $(q_I, w, Z_0) \vdash_{\mathsf{M}}^* (q_I, \epsilon, X)$ for some X with $|X| = numsep(w)$.

Theorem 5. *Let Q_1 and Q_2 be quantifiers corresponding to regular languages. The language accepted by $\mathsf{It}(\mathsf{Q}_1, \mathsf{Q}_2)$ is $\mathcal{L}_{Q_1 \cdot Q_2}$.*

Proof. First, $\mathcal{L}_{Q_1 \cdot Q_2} \subseteq L\left(\mathsf{It}\left(\mathsf{Q}_1, \mathsf{Q}_2\right)\right)$. In particular, we show by induction on $n = \mathit{numsep}\left(w\right)$ for $w \in \mathcal{L}_{Q_1 \cdot Q_2}$ that $\left(q_I, w, Z_0\right) \vdash^*_{\mathsf{M}} \left(q_I, \epsilon, X\right)$ where

$$\begin{aligned} \operatorname{card}\left(\left\{w_i \mid w_i \in \mathcal{L}_{Q_2}\right\}\right) &= \#_1\left(X\right) \\ \operatorname{card}\left(\left\{w_i \mid w_i \notin \mathcal{L}_{Q_2}\right\}\right) &= \#_0\left(X\right) \end{aligned}$$

The inclusion then follows immediately from the definition of the pushdown reader Q^p_1 and the $\epsilon, x/x$ transition from q_I to $q_0\left(\mathsf{Q}^p_1\right)$. If $n = 0$, then $w = \epsilon$. This is in $\mathcal{L}_{Q_1 \cdot Q_2}$ only if $\langle 0, 0\rangle \in Q^c_1$. One can easily check that this entails $q_0\left(\mathsf{Q}_1\right) \in F\left(\mathsf{Q}_1\right)$, so the ϵ transition from q_I to $q_0\left(\mathsf{Q}^p_1\right)$ takes us to an accepting state of M.

For the inductive step, assume $n > 0$. Write $w = w^- w_i \boxdot$. By inductive hypothesis, we have a w^- path from q_I to q_I generating a stack X such that

$$\begin{aligned} \operatorname{card}\left(\left\{w_i \mid w_i \in w^- \text{ and } w_i \in \mathcal{L}_{Q_2}\right\}\right) &= \#_1\left(X\right) \\ \operatorname{card}\left(\left\{w_i \mid w_i \in w^- \text{ and } w_i \notin \mathcal{L}_{Q_2}\right\}\right) &= \#_0\left(X\right) \end{aligned}$$

By Lemma 3, there is a w sequence from q_I to q_I which generates a stack $1X$ or $0X$ depending on whether $w_i \in \mathcal{L}_{Q_2}$ or not, exactly as desired.

For the $\mathcal{L}_{Q_1 \cdot Q_2} \supseteq L\left(\mathsf{It}\left(\mathsf{Q}_1, \mathsf{Q}_2\right)\right)$ inclusion, consider $w \in L\left(\mathsf{M}\right)$. Because $F(\mathsf{M}) = F\left(\mathsf{Q}^p_1\right)$ and Q^p_1 only pops from the stack, there must be a w sequence from q_I to q_I generating a stack that contains a word accepted by Q_1. Because the only transitions leaving q_I are ε and the only ones back to q_I are $\boxdot, x/x$, w must be of the form $\left(w_i \boxdot\right)^*$. That $w \in \mathcal{L}^n_{Q_1 \cdot Q_2}$ then follows by $\mathit{numsep}(w)$ applications of Lemma 3 and by inspection of Definition 9.

Although we have demonstrated that languages for iterating two quantifiers whose languages can be processed by finite state automata can be processed by pushdown automata, a natural question arises: can these iterated languages also be accepted by finite state automata? In other words, are the regular languages closed under iteration? We show the answer to be positive.

First, note the following fact about regular expressions.[15]

Lemma 4. *If E is a regular expression in alphabet $\{a_1, \ldots, a_n\}$ and E_i is a regular expression in alphabet Σ_i, then $E\left[a_1/E_1, \ldots, a_n/E_n\right]$ is a regular expression in $\bigcup_{i \leqslant n} \Sigma_i$.*

Definition 12. Let E_1 and E_2 be regular expressions in $\{0, 1\}$. We define the iterated regular expressions $\mathit{It}\left(E_1, E_2\right)$ by

$$\mathit{It}\left(E_1, E_2\right) = E_1\left[0/\left(E^c_2 \boxdot\right), 1/\left(E_2 \boxdot\right)\right]$$

where E^c denotes a regular expression for the complement of the language generated by E (recall the regular languages are closed under complement).

[15]In fact, the substitutions we are defining are known in the compilers literature as *regular definitions*. See, for instance, [Aho et al., 2006, Ch. 3].

From Lemma 4 we know that $It\,(E_1, E_2)$ is a regular expression for every n. Inspection of the above definition makes the following closure result obvious.

Proposition 2. *Let Q_1 and Q_2 be quantifiers with regular languages; write E_{Q_1} and E_{Q_2} for a regular expression generating $\mathcal{L}_{Q_1}$ and $\mathcal{L}_{Q_2}$. Then $It\,(E_{Q_1}, E_{Q_2})$ generates $\mathcal{L}_{Q_1 \cdot Q_2}$. In other words, $\mathcal{L}_{Q_1 \cdot Q_2}$ is a regular language whenever both $\mathcal{L}_{Q_1}$ and $\mathcal{L}_{Q_2}$ are.*

5.1.1 Note on Processing

Already in the single quantifier case, certain quantifiers like *an even number of* have both DFA and PDA representations. It has been suggested, with supporting evidence (Szymanik and Zajenkowski [2010a]), that working memory is solicited when processing sentences containing such quantifiers. This provides *prima facie* reason to believe that working memory will be recruited when processing sentences with multiple quantifiers each computable by a DFA. This would show that the PDA representation more closely resembles the actual processing mechanism.

One argument in support of the DFA representation could derive from Fact 1 about the size of the PDAs. Both $\mathsf{It}\,(\mathsf{some}, \mathsf{every})$ and $\mathsf{It}\,(\mathsf{every}, \mathsf{some})$ have five states. It can be shown, however, that the minimal DFA accepting $\mathcal{L}_{some \cdot every}$ has four states and that the minimal DFA accepting $\mathcal{L}_{every \cdot some}$, depicted in Figure 6, has three states, strictly fewer than the associated PDAs.

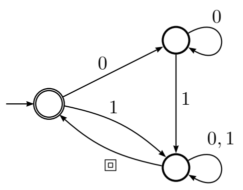


Figure 6: The minimal DFA accepting the language for *every A R some B*.

This smaller state space and the apparent superfluity of the stack may favor the DFA representation for such iterations. On the other hand, the PDA construction provides a general method for generating a machine for the iteration of any two quantifiers. There appears to be no such analogously general mechanism for generating the minimal DFAs. Because neither argument on its own can be conclusive, empirical investigation should be done to see how much (if any) working memory is activated in these and similar cases.

5.2 Iterating with One or More Pushdown Automata

We now consider the case where one or both of the quantifiers in the iteration defines a non-regular, context-free language. Note that Definition 8 and Proposition 1 still apply in this situation. To define machines accepting these iterated languages, we proceed as before by adding a stack to act as an input tape to a pushdown reader. Because one or both of the machines being iterated may in fact be a pushdown automaton, we must generate a two-stack pushdown automaton.

Definition 13. A *two-stack pushdown automaton* is exactly like a pushdown automaton except that the transition function is now of the form

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times \Gamma \to \mathcal{P}(Q \times \Gamma^* \times \Gamma^*)$$

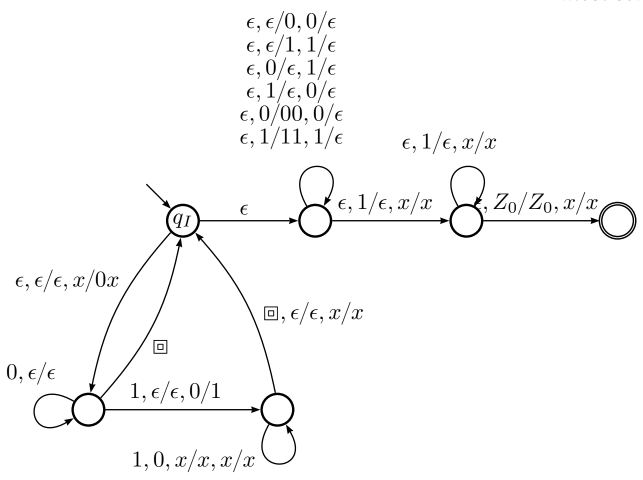
We depict a transition $\delta(q, a, X_1, X_2) = (p, \gamma_1, \gamma_2)$ by a directed arc from q to p, labeled by $a, X_1/\gamma_1, X_2/\gamma_2$.

Definition 14. Let M be a PDA. The *pushdown reader of* M, M^p, is a two-stack pushdown automaton defined by

- $Q(\mathsf{M}^p) = Q(\mathsf{M})$, $q_0(\mathsf{M}^p) = q_0$, $F(\mathsf{M}^p) = F(\mathsf{M})$
- $\Sigma(\mathsf{M}^p) = \varnothing$
- $\Gamma(\mathsf{M}^p) = \Sigma(\mathsf{M})$
- $\delta(\mathsf{M}^p) = \{\langle q_1, \epsilon, X, r, q_2, \gamma, \epsilon\rangle \mid \langle q_1, r, X, q_2, \gamma\rangle \in \delta(\mathsf{M})\}$

In the pushdown reader, all $r, X/\gamma$ transitions in the original machine become $\epsilon, X/\gamma, r/\epsilon$ transitions. Definition 10 of iterated machines easily generalizes when one (or both) of Q_1 and Q_2 is a PDA. The construction is nearly identical, merely keeping track of the extra stack. An example will make this clearer.

Example 3. Figure 7 depicts a two-stack PDA accepting $\mathcal{L}^n_{most \cdot some}$.



$\epsilon, \epsilon/0, 0/\epsilon$
$\epsilon, \epsilon/1, 1/\epsilon$
$\epsilon, 0/\epsilon, 1/\epsilon$
$\epsilon, 1/\epsilon, 0/\epsilon$
$\epsilon, 0/00, 0/\epsilon$
$\epsilon, 1/11, 1/\epsilon$
$\epsilon, 1/\epsilon, x/x$
q_I
ϵ
$\epsilon, 1/\epsilon, x/x$
$\epsilon, Z_0/Z_0, x/x$
$\epsilon, \epsilon/\epsilon, x/0x$
$\boxdot$
$\boxdot, \epsilon/\epsilon, x/x$
$0, \epsilon/\epsilon$
$1, \epsilon/\epsilon, 0/1$
$1, 0, x/x, x/x$

Figure 7: A pushdown automaton for *most A R some B*.

As a model of computation, two-stack PDAs have the same power as Turing machines, in that every Turing machine can be simulated by a two-stack PDA and *vice versa*. One might wonder whether our machines really need the power of two stacks or whether, as was the case with the regular languages, the added stack can be eliminated while accepting the same language. In other words, the question arises of whether the context-free languages are closed under iteration (both with regular languages and with other context-free languages).

Again, the answer is positive. The approach directly mirrors that for showing closure of regular languages, replacing 1s and 0s in the language of Q_1 by words in the (appropriately indexed) language of Q_2. First, a simple observation:

Fact 2. For a quantifier Q, if $\mathcal{L}_Q$ is context-free, then so too is $\mathcal{L}_{\neg Q}$.

Proof. Let φ be a formula in first-order additive arithmetic defining Q (as given by Theorem 4). Then $\neg\varphi$ defines $\neg Q$.

Theorem 6. *If $\mathcal{L}_{Q_1}$ and $\mathcal{L}_{Q_2}$ are context-free, then so is $\mathcal{L}_{Q_1 \cdot Q_2}$.*

Proof. Let G_1, G_2, and $\overline{G_2}$ be context-free grammars (CFGs) in alphabet $\{0,1\}$ generating $\mathcal{L}_{Q_1}$, $\mathcal{L}_{Q_2}$, and $\mathcal{L}_{\neg Q_2}$ respectively. Let $n > 0$. We will construct a CFG G in alphabet $\{0, 1, \boxdot\}$ generating $\mathcal{L}_{Q_1 \cdot Q_2}$.

The start symbol of G is the start symbol of G_1. We add a copy of G_2 and call its start symbol S_2. Similarly for $\overline{G_2}$. We then have two new production rules:

$$\begin{aligned} W &\to S_2 \boxdot \\ \overline{W} &\to \overline{S_2} \end{aligned}$$

G simply contains the rules of $G_1\left[0/\overline{W}, 1/W\right]$ and the production rules outlined in the previous paragraph. That G generates $\mathcal{L}_{Q_1 \cdot Q_2}$ follows *via* an analogous argument to that given in the proof of Proposition 2; intuitively, W generates some $w_i \in \mathcal{L}_{Q_2}$ and $\overline{W}$ some $w \in \mathcal{L}^i_{\neg Q_2}$.

Example 4. The grammar below generates exactly $\mathcal{L}^n_{most \cdot some}$:

$$\begin{aligned} M &\to MWM \mid M'WM \mid MWM' \mid M'WM' \\ M' &\to WM'\overline{W} \mid \overline{W}M'W \mid \varepsilon \\ W &\to S_2 \boxdot \\ \overline{W} &\to \overline{S_2} \boxdot \\ S_2 &\to S_2' 1 S_2' \\ S_2' &\to S_2' S_2' \mid 1 \mid 0 \mid \varepsilon \\ \overline{S_2} &\to 0\overline{S_2} \mid \varepsilon \end{aligned}$$

Just as one can algorithmically generate a DFA from a regular expression, so too can a PDA be generated from a CFG. There is not, however, an analog of the minimal DFA in the case of PDAs. It is also less clear what repercussions this closure result would have for issues of language processing.

6 Conclusion

Our main contribution in this paper is the first extension of the semantic automata framework to polyadic quantification. After presenting the classical results in this area, and discussing general issues in the connection between automata and processing, we showed how to model sentences with iterated quantification. We also showed that the regular and context-free languages are closed under iteration in a precise sense. This is the first step in gaining a better understanding of how these machine models might relate more generally to uses of quantifiers in natural language. At this point a number of new questions suggestion themselves for further investigation. Some of these include:

- The extension of semantic automata to iterated quantification gives rise to new empirical predictions, as mentioned above. These predictions should be tested, and more detailed predictions should be explored.
- In light of the discussion in Section 4, it is easy to imagine probabilistic automata or other extensions, reflecting either biases or specific algorithmic verification strategies. This could allow more detailed process models that could be subject to more precise behavioral experimentation.
- Having defined automata for iteration, it is natural to consider other polyadic lifts: resumption, cumulation, branching.[16]
- It may be possible to define automata for irreducibly polyadic quantifiers, which could allow another angle on understanding the elusive Frege boundary (van Benthem [1989], Keenan [1992]), through semantic automata.
- A related theoretical question concerns whether minimal DFAs for iterated quantifiers may or must contain non-trivial cycles. This can be phrased more precisely by asking whether these regular languages have non-zero star height (see McNaughton and Papert [1971]).
- The close relationship between quantifiers and formal languages in the semantic automata framework allows some results from mathematical linguistics to be used to address semantic issues. For instance, semantic learning may be study in the context of the learnability in the limit framework (Gold [1967]). First steps in this direction have been taken by Gierasimczuk [2009] and Clark [2011].

We hope to pursue these questions in future work.

References

Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Prentice Hall, 2nd edition, 2006.

[16] As a reviewer pointed out, we can already define a machine for cumulation as the sequential composition of $\mathsf{It}\,(\mathsf{Q}_1, \mathsf{some})$ and $\mathsf{It}\,(\mathsf{Q}_2, \mathsf{some})$.

Jon Barwise and Robin Cooper. Generalized Quantifiers and Natural Language. *Linguistics and Philosophy*, 4(2):159–219, 1981.

Johan van Benthem. *Essays in Logical Semantics*. D. Reidel Publishing Company, Dordrecht, 1986.

Johan van Benthem. Polyadic quantifiers. *Linguistics and Philosophy*, 12(4): 437–464, August 1989. ISSN 0165-0157. doi: 10.1007/BF00632472. URL `http://www.springerlink.com/index/10.1007/BF00632472`.

Nick Chater and Mike Oaksford. The Probability Heuristics Model of Syllogistic Reasoning. *Cognitive Psychology*, 38(2):191–258, March 1999. ISSN 0010-0285. doi: 10.1006/cogp.1998.0696. URL `http://www.ncbi.nlm.nih.gov/pubmed/10090803`.

Noam Chomsky. On Certain Formal Properties of Grammars. *Information and Control*, 2(2):137–167, June 1959. ISSN 00199958. doi: 10.1016/S0019-9958(59)90362-6. URL `http://linkinghub.elsevier.com/retrieve/pii/S0019995859903626`.

Robin Clark. On the Learnability of Quantifiers. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 911–923. Elsevier B.V., second edition, 2011. ISBN 9780444537263. doi: 10.1016/B978-0-444-53726-3.00020-7. URL `http://dx.doi.org/10.1016/B978-0-444-53726-3.00020-7`.

Stanislas Dehaene. *The Number Sense: How the Mind Creates Mathematics*. Oxford University Press, Oxford, 1997.

Nina Gierasimczuk. Identification through Inductive Verification Application to Monotone Quantifiers. In P Bosch, D Gabelaia, and J Lang, editors, *7th International Tbilsi Symposium on Logic, Language, and Computation, TbiLLC 2007*, volume 5422 of *Lecture Notes in Artificial Intelligence*, pages 193–205. 2009.

E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, May 1967. ISSN 00199958. doi: 10.1016/S0019-9958(67)91165-5. URL `http://linkinghub.elsevier.com/retrieve/pii/S0019995867911655`.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, Boston, 2nd edition, 2001.

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

Edward L. Keenan. The Semantics of Determiners. In Shalom Lappin, editor, *The Handbook of Contermporary Semantic Theory*, number 1989, chapter 2, pages 41–63. Blackwell, Oxford, 1996.

Edward L. Keenan and Dag Westerståhl. Generalized Quantifiers in Linguistics and Logic. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 859–910. Elsevier, second edition, 2011.

David Lewis. General semantics. *Synthese*, 22:18–67, 1970.

Jeffrey Lidz, Paul Pietroski, Justin Halberda, and Tim Hunter. Interface transparency and the psychosemantics of most. *Natural Language Semantics*, 19(3): 227–256, April 2011. ISSN 0925-854X. doi: 10.1007/s11050-010-9062-6. URL `http://www.springerlink.com/index/10.1007/s11050-010-9062-6`.

Per Lindström. First order predicate logic with generalized quantifiers. *Theoria*, 32:186–195, December 1966. ISSN 00224812. URL `http://onlinelibrary.wiley.com/doi/10.1111/j.1755-2567.1966.tb00600.x/abstract`.

David Marr. *Vision.* Freeman, San Francisco, 1982.

Corey T. McMillan, Robin Clark, Peachie Moore, Christian Devita, and Murray Grossman. Neural basis for generalized quantifier comprehension. *Neuropsychologia*, 43(12):1729–1737, January 2005. ISSN 0028-3932. doi: 10.1016/j.neuropsychologia.2005.02.012. URL `http://www.ncbi.nlm.nih.gov/pubmed/16154448`.

Corey T. McMillan, Robin Clark, Peachie Moore, and Murray Grossman. Quantifier comprehension in corticobasal degeneration. *Brain and Cognition*, 62(3): 250–260, December 2006. ISSN 0278-2626. doi: 10.1016/j.bandc.2006.06.005. URL `http://www.ncbi.nlm.nih.gov/pubmed/16949714`.

Robert McNaughton and Seymour A. Papert. *Counter-free Automata*, volume 65 of *MIT Research Monographs.* The MIT Press, 1971.

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

Marcin Mostowski. Divisibility Quantifiers. *Bulletin of the Section of Logic*, 20 (2):67–70, 1991.

Marcin Mostowski. Computational semantics for monadic quantifiers. *Journal of Applied Non-Classical Logics*, 8:107–121, 1998.

Christopher Peacocke. Explanation in Computational Psychology: Language, Perception and Level 1.5. *Mind and Language*, 1(2):101–123, 1986.

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

Paul Pietroski, Jeffrey Lidz, Tim Hunter, and Justin Halberda. The Meaning of 'Most': Semantics, Numerosity and Psychology. *Mind and Language*, 24 (5):554–585, 2009.

Paul Smolensky. On the proper treament of connectionism. *Behavioral and Brain Sciences*, 11:1–23, 1988.

Scott Soames. Linguistics and Psychology. *Linguistics and Philosophy*, 7(2): 155–179, 1984.

Patrick Suppes. Procedural Semantics. In R Haller and W Grassl, editors, *Language, Logic, and Philosophy: Proceedings of the 4th International Wittgenstein Symposium*, pages 27–35. Hölder-Pichler-Tempsy, Vienna, 1980.

Anna Szabolcsi. *Quantification.* Research Surveys in Linguistics. Cambridge University Press, Cambridge, 2009.

Jakub Szymanik. A comment on a neuroimaging study of natural language quantifier comprehension. *Neuropsychologia*, 45(9):2158–2160, May 2007. ISSN 0028-3932. doi: 10.1016/j.neuropsychologia.2007.01.016. URL `http://www.ncbi.nlm.nih.gov/pubmed/17336347`.

Jakub Szymanik. Computational complexity of polyadic lifts of generalized quantifiers in natural language. *Linguistics and Philosophy*, 33(3):215–250, November 2010. ISSN 0165-0157. doi: 10.1007/s10988-010-9076-z. URL `http://www.springerlink.com/index/10.1007/s10988-010-9076-z`.

Jakub Szymanik and Marcin Zajenkowski. Comprehension of simple quantifiers: empirical evaluation of a computational model. *Cognitive Science*, 34(3):521–532, April 2010a. ISSN 1551-6709. doi: 10.1111/j.1551-6709.2009.01078.x. URL `http://www.ncbi.nlm.nih.gov/pubmed/21564222`.

Jakub Szymanik and Marcin Zajenkowski. Quantifiers and Working Memory. *Lecture Notes in Artificial Intelligence*, 6042:456–464, 2010b.

Jakub Szymanik and Marcin Zajenkowski. Contribution of working memory in parity and proportional judgments. *Belgian Journal of Linguistics*, 25(1):176–194, January 2011. ISSN 07745141. doi: 10.1075/bjl.25.08szy. URL `http://openurl.ingenta.com/content/xref?genre=article&issn=0774-5141&volume=25&issue=1&spage=176`.

Dag Westerståhl. Quantifiers in Formal and Natural Languages. In D Gabbay and F Guenthner, editors, *Handbook of Philosophical Logic (vol. 4)*, pages 1–132. D. Reidel Publishing Company, Dordrecht, 1989.