Human and Artificial Intelligence Acquisition of Quantifiers

by

Samson S. Zhou

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2011

Author ....
Department of Electrical Engineering and Computer Science
August 22, 2011

Certified by............ [illegible]
Prof. Robert C. Berwick
Professor of Computational Linguistics, Department of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by.. [illegible]
Prof. Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

Human and Artificial Intelligence Acquisition of Quantifiers

by

Samson S. Zhou

Submitted to the Department of Electrical Engineering and Computer Science
on August 22, 2011, in partial fulfillment of the
requirements for the degree of
Masters of Science in Computer Science and Engineering

Abstract

This paper is concerned with constraints on learning quantifiers, particularly those cognitive on human learning and algorithmic on machine learning, and the resulting implications of those constraints on language identification. Previous experiments show that children attempting to differentiate quantifiers from numbers use a similar acquisition method for both types of words. However, some types of natural quantifiers, such as all but do not appear as a single word in any human language, perhaps due to either what would be an ineffective definition, or due to what would seem to be an unnatural definition. On the other hand, the constraints of language acquisition by identification place strong constraints on possible languages to identify an unknown language in a certain given class of languages. The experiment presented in this paper measures the cognitive ability of humans to acquire quantifiers, both conservative and non-conservative, through a series of positive and negative training examples. It then implements an algorithm used to acquired quantifiers which can be expressed as regular languages in the minimal number of states in its determinate finite automata representation in polynomial time.

Thesis Supervisor: Prof. Robert C. Berwick
Title: Professor of Computational Linguistics, Department of Electrical Engineering and Computer Science

Acknowledgments

This paper would not have been possible without the support, advice, and encouragement of several individuals, who in one way or another contributed to the success and the completion of this project.

First and foremost, I would like to extend my utmost gratitude to my thesis supervisor, Professor Robert C. Berwick, who provided unfailing knowledge and keen guidance and enabled me a deeper understanding of the subject. I sincerely appreciate the patience and accommodation exhibited by Professor Berwick in the early stages of my work and the valuable insight provided throughout the study. I am thankful for the excellent example he set for academic excellence and am greatly indebted for my growth as a researcher.

To Peter Graff, my Undergraduate Research Opportunities Program supervisor, I am grateful for the advice and direction he provided throughout the duration of the program. Without his interest in this subject, I would have never developed as a scientist in this field.

To Anne Hunter, Undergraduate Administrator for the Computer Science and Electrical Engineering Department, I would like to sincerely thank, for her everlasting support and accessibility. Her knowledge allowed me to surpass many obstacles in the completion of this paper. It was truly a pleasure to work with such an individual.

This experience with this project and the Massachusetts Institute of Technology has been made enjoyable by the friends who became an integral part of my life as I completed my degree. I would like to thank David Kelley, Elena Tatarchenko, Jennifer Li, and Joseph Laurendi for their support and expertise in the final stages of writing this paper. Additionally, I appreciate the support displayed by David Lee and Mark Zhang, encouraging me to pursue this field at MIT. I am also grateful for the time spent with members of the summer Ultimate Frisbee BUDA team, Brendan Lundy, Diana Kwan, Edward Wong, Lisa Liu, Oghosa Ohiomoba, Shawn Le, and Stephen

Lo, for their inspirational camaraderie and companionship. I would like to thank my fellow members of Zeta Beta Tau fraternity who persisted by my side in solidarity, especially Daniel Gerber, Erik Feng, Jeffrey Xing, Harley Zhang, Matthew Vaughan, Rajeev Nayak, and Stephen Tsai.

Lastly, I would like to thank my family members, for their unwavering support and steadfast encouragement. Without them, I would have never been here in the first place, and I am deeply grateful for everything they have done for me in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The motivation to study this particular problem arises from the fields of computational linguistics and psycholinguistics. For most languages, it is infeasible or even impossible to compile a complete set of governing rules to teach learning systems. Similarly, learning systems cannot feasibly contain all possible solutions for an unknown definition, and must be trained appropriately so that they can adapt to new definitions. Although linguistics attempts to understand how children acquire language, this field attempts to understand how computers learn language. Recreating a Bayesian learning program to adapt to non-conservative quantifiers also serves as an interesting challenge which will further understanding of existing theory. Additionally, discovering the limitations of a previously published algorithm explores the structure of the algorithm so that possible optimizations can either enlarge the set of inputs or expedite the evaluation process. A successful implementation of a language identification artificial intelligence would shed significant light on information regarding the structure of human language.

Chapter 2

Human Acquisition

2.1 Introduction to Quantifiers

A quantifier in human language is a modifier that describes quantity. For example, in the statement "all of the students are girls", the word *all* is a quantifier modifying the set of *students.* Other quantifiers in the English language include *every*, *some*, and *most.* However, quantifiers can be syntactically very complex, as there is no way to express the same idea as a conjunction or disjunction of sentences, each of the form "that student is a girl". Those quantifiers listed above are among the conservative quantifiers that are present in the English language. And while every language make use of quantifiers, no language represents non-conservative quantifiers, such as "all of the objects that are NOT circles" as a single word.

2.2 Generalized Quantifiers

Although quantifiers such as "all of" and "one of" are very common pragmatically, combinations of logical operators open the possibilities of many more quantifiers. In linguistic semantics, a generalized quantifier describes a higher order property, that is, the property of a property. So, instead of saying "all students are girls", a generalized quantifier would say that "all students are students who are girls". Mathematically, the generalized quantifier would be formulated:

$$\{x|x \textbf{ is a student}\} \in \{x|x \textbf{ is a girl}\}$$

Generalized quantifiers can have properties such as monotonicity, but the relevant property here is conservativity. A generalized quantifier Q is said to be *conservative* if

$$Q(A)(B) \Leftrightarrow Q(A)(A \cap B).$$

As stated above, the following two statements are equivalent:

1. All of the students are girls.

2. All of the students are students who are girls.

On the other hand, the word *only*, which is not usually treated as a quantifier, is not conservative, as the following two statements are not equivalent:

1. Only the students are girls.

2. Only the students are students who are girls.

2.3 Acquisition of Quantifiers

An interesting experiment involves the acquisition of quantifiers and number terms for young children. Because both number words and quantifiers involve the concept of multiplicity, young children apply the same syntactical, pragmatic, and semantic approach in learning these ideas. In the English syntax, numbers and quantifiers both occur before adjective modifiers and have count syntax (two/some geese). Moreover, both numbers and quantifiers can co-exist with partitives (two/some of the balls). Semantically, both numerals and quantifiers are predicates over sets of individuals. Furthermore, both numbers and quantifiers form an order from weaker to stronger entities within each set. For example, in the number scale, three is always greater than two, which is always greater than one. Under the quantifier scale, all is always greater than most, which is greater than many, and so forth.

Number order: $\ldots > 3 > 2 > 1$

Quantifier order: all $>$ most $>$ many $>$ some $>$ none

However, traditional usage of the above terms also have implied lexical bounds. For example, the statement "two items are good" logically means that at least two, perhaps more, items are good. Similarly the statement "some items are good" means that at least some, perhaps all, of the items are good. However, using a number or quantifier with a small order usually excludes a higher ranked member of that same scale in the practical sense. English speakers generally say "some of the items" to mean that not all of the items are good. These colloquial inferences, called scalar implications, result from the conversational presumption that speakers should be informative, but clear. Therefore, using a lower ranked number or quantifier often excludes the application of a higher ranked number of quantifier. Thus, for example, if told "Barack Obama has one leg", all competent users of English will conclude, that Barack Obama does not have two legs. Listeners who hear "some Republicans voted for Obama" will conclude that not all Republicans voted for "Obama".

2.4 Quantifier Comprehension and Verification

Although quantifiers all express some implicit numeric property of the specified item, one would suspect there is also some scale of complexity about them. In fact, even the subtle substitution of two quantifiers with the same definition could change perception of these concepts. For example, while the quantifiers "most" and "more than half" always produce the same results in verifications, might the slight difference in semantics cause slightly different verification process?

1. "Most"(A)(B) $\Leftrightarrow |A \cap B| > |A - B|$

Table 2.1: Mean reaction time and standard deviation in response times

Group	Quantifiers	M	SD
Aristotelian FO	all, some	2257	471.95
Parity	even, odd	5751	1240.41
Cardinal FO	less than eight, more than seven	6035	1071.89
Proportional	less than half, more than half	7273	1410.48

2. "More than half"(A)(B) $\Leftrightarrow |A \cap B| > \frac{1}{2}|A|$

In previous experiments any differences in accuracy and response time in verification for the two quantifiers were not statistically significant. [6] Similarly, nothing significant was determined in differentiating the quantifiers "at least n" versus "more than $n + 1$".

However, quantifiers with different meanings likely differ in verification time, largely dependent on the complexity of these quantifiers. For example, being able to deduce whether "most of the items are good" will likely be more difficult than being able to deduce whether "none of the items are good". To thoroughly examine the above claim, a previous experiment separated quantifiers into three categories, those recognized by acyclic finite automata, those recognized by general finite automata, and those recognized by push-down automata, and compared human reaction times needed for these classes of quantifiers. Participants in the experiment were given a simple picture with fifteen cars and a corresponding statement such as "less than half of the cars are blue", the validity of which the participants needed to ascertain. The table [8] summarizing the results is below with mean reaction time (M) and standard deviation (SD) in milliseconds. Even though Aristotelian and parity quantifiers are both recognized by finite automata, there is a noticeable difference between the two, perhaps due to the fact that the Aristotelian quantifiers can be represented by acyclic finite automata, whereas finite automata representing parity quantifiers require loops. This project will involve a similar idea with non-conservative quantifiers also included, to see exactly what computational complexity the human mind incorporates.

Chapter 3

Artificial Intelligence Acquisition

3.1 Bayesian Learning of Quantifiers

Combining the ideas of quantifiers and machine learning, attempts to formulate algorithms that can learn the definition of an unknown quantifiers have also been developed. One such experiment used a standard Bayesian learning technique to assign probabilities to each possible quantifier in the solution set. After a series of training examples, the learning system tries to determine which quantifier best matches the unknown quantifier. The experiment concluded that even with as few as four examples, a reliable learning system can be trained to learn the meaning of any first-order quantifier and answer queries about an unknown quantifier. The choice of the training examples also influenced the success rate of the learning system. With a series of non-random training examples, the outcome can actually be improved, especially if the training examples are near the boundary of valid examples. Also, the learning system adapted the property that increases weight to quantifiers that are learned well, and decreases weight to those that are not frequent. This project will attempt to recreate this model and apply it to cases where the unknown quantifier is non-conservative. The resulting analysis will provide control data as to whether the concept of non-conservative quantifiers is computationally complex, or if there exists cognitive constraints on human learning. While the first part of the proposed project will focus on analyzing human acquisition of both conservative and non-conservative quantifiers and comparing those results to algorithmic learning of the same concepts, the second part of the proposed project will look at language identification algorithms.

3.2 Introduction to Language Identification

The question of language identification is a challenging subject which lends itself to complicated analysis. The overarching problem is whether a program can identify whether or not some input is a member of a larger group. More specifically, a *language* is a set of strings on some predetermined finite alphabet. The program is then given some information on the rules of that language, often in the form of training examples. That is, the program can be taught which strings exist in this language, and/or which strings are not members of this language. While the algorithm attempting to identify the language must rely strictly on given information, the provided information must be complete enough so that some program can theoretically acquire the rules of the selected language. That is, there should be enough examples and counterexamples to clarify the intricacies of the language. For example, if an artificial intelligence were attempting to identify the English language, possible given information can either be a comprehensive list of correct English usage or a comprehensive list of correct and incorrect English usage, with each item in the list given as either "correct" or "incorrect".

Although there exist many different success conditions, under this paper, the goal of the artificial intelligence is to identify in the limit a class of languages. That is, time is finite and discretized. At each time $t = 1, 2, \dots,$ the learner is given a unit of information i_t regarding the unknown language L. The learner must then make a guess g_t as to which language L might be based ont eh cumulative information given. Then L is said to be *identified in the limit* if, after some finite time, the guesses are all the same, and correctly identified as L. The learner will never know that the guess is actually correct, because there always exists the possibility that the next unit of information will discount the previous guess. However, the learner must continue returning the correct guess following each unit of information. By extension, a class of languages is *identifiable in the limit* if all languages in that class are identifiable in the limit by the same computable function.

3.3 Text

As previously stated, the information given to the learner must suffice so that the learner can theoretically identify the language. This paper address two fundamental sources of information. The first source, a *text* for a language L is a sequence of strings s_q, s_2, ... from L such that every string of L must occur at least once in the sequence. Obviously, many such sequences can exist given a language L. However, this paper specifies three types of text:

- *Arbitrary Text*: Each string s_t in the sequence can come from any function of t.
- *Recursive Text*: Each string s_t in the sequence must come from a recursive function of t.
- *Primitive Recursive Text*: Each string s_t in the sequence must come from a primitive recursive function of t.

3.4 Informant

Another type of information presentation is an *informant*, which can tell the learner whether a given string s_t is in L or not, but it may not divest the rules which it uses to decide this. There are also three interesting types of informants:

- *Arbitrary Informant*: Each string s_t in the sequence can come from any function of t as long as every string of L occurs at least once.
- *Methodical Informant*: Each string s_t in the sequence is assigned a priori.
- *Request Informant*: At any time step t, the learner can choose s_t depending on the previous information received.

The distinction between text and informant, is that text only presents positive information, examples of correct sentences, but no examples of incorrect sentences. Informants, however, present negative information. For the time being, the information provided is limited to either text or informant. In the human learning process, instances of incorrect usages are often not corrected, and even those that are corrected may still resurface incorrectly in the future. Therefore, it might be reasonable to theorize that language can be identified with only positive examples, such as a text. However, previous results have shown that only trivial classes of languages are identifiable in the limit given only text and attempts to construct a learner which could come to meaningful conclusions given only text have not been successful. Hence, one must wonder if there is even enough information in a text to identify a context-free language.

3.5 A Learning Algorithm

An algorithm, first described and annotated by Angluin [1] and later summarized by Clark [3], efficiently learns an initially unknown regular language from membership queries and examples. Because regular languages are accepted by deterministic finite state machines, this process by which the learner acquires new definitions is illustrated with examples in the form of finite state machines.

3.5.1 Notation

Let the unknown regular set be U over known alphabet A, which is fixed and finite. Throughout the process, the algorithm maintains and updates an observation table, its legend for the finite state machine. The observation table, denoted (S, E, T) consists of three things:

1. A non-empty prefix closed set S of strings, where every prefix of every member in the set is also a member of the set. As per the notation in Clark, a string ω is a prefix of another string π if and only if there exists a string ρ such that

$\pi = \omega \cdot \rho$, where the $\cdot$ is the concatenation function.

2. A non-empty suffix closed set E of strings, where suffix closed sets are defined analogously to prefix closed sets.

3. A finite function T mapping $((S \cup S \cdot A) \cdot E)$ to $\{0,1\}$, where the output is 1 if and only if the string is in the regular set.

Therefore, the observation table is a two-dimensional array with columns labeled by elements of E, rows labeled by elements of S, and each cell in row s and column e containing the value of $T(s \cdot e)$. Then define $row(s)$ to be the row labeled by element s.

As the algorithm proceeds, the learner repeatedly makes conjectures to a request informant. However, this type of request informant, the *minimally adequate informant*, takes conjectures of the regular set. The informant returns *yes* if the conjecture is correct. Otherwise, it returns a counterexample which is in the symmetric difference of the conjecture and the unknown regular language. However, there is no restriction on which counterexample the minimally adequate teacher can return. Meanwhile, the learner updates the observation table based upon the queries to the teacher, while trying to keep the table both closed and consistent, where

- An observation table is consistent if and only if for each $s_1, s_2 \in S$ such that $row(s_1) = row(s_2)$, then for all $a \in A$, $row(s_1 \cdot a) = row(s_2 \cdot a)$.

- An observation table is closed if and only if for each $t \in S \cdot A$, there exists $s \in S$ such that $row(s) = row(t)$.

Note that an acceptor in the form of a finite state machine can now be formulated as $M(S, E, T) = (Q, \sum, \delta, q_0, F)$ over alphabet A where

- $Q = \{row(s) | \, s \in S\}$ is the state set

- $q_0 = row(\lambda)$ is the initial state

- $F = \{row(s) | \, s \in S, T(s) = 1\}$ is the accepting states

- $\delta(row(s), a) = row(s \cdot a)$ is the transition function

3.5.2 Algorithm

The premise of the algorithm relies on the observation table to maintain knowledge accrued throughout the process while repeatedly querying the minimally adequate teacher after failed conjectures. The pseudocode from [1] follows:

```
Initialize $S$ and $E$ to $\{\lambda\}$.
Ask membership queries for $\lambda$ and each $a \in A$.
Construct the initial observation table $(S,E,T)$.

Repeat:
  While $(S,E,T)$ is not closed or not consistent:
    If $(S,E,T)$ is not consistent:
      Find $s_1, s_2 \in S, a \in A, e \in E$ such that $row(s_1) = row(s_2)$
        but $row(s_1 \cdot a \cdot e) \neq row(s_2 \cdot a \cdot e)$
      Add $a \cdot e$ to $E$
      Extend $T$ to $(S \cup S \cdot A) \cdot E$ using membership queries.
    If $(S,E,T)$ is not closed,
      Find $s_1 \in S, a \in A$ such that $row(s_1 \cdot a)$ is different
        from $row(s)$ for all $s \in S$
      Add $s_1 \cdot a$ to $S$
      Extend $T$ to $(S \cup S \cdot A) \cdot E$ using membership queries
  If $(S,E,T)$ is closed and consistent, define $M = M(S,E,T)$
  Make conjecture $M$ to the teacher
  If teacher gives counterexample $t$
    Add $t$ and its prefixes to $S$
    Extend $T$ to $(S \cup S \cdot A) \cdot E$ using membership queries
If teacher replies ''yes'' to the conjecture, output $M$.
```

In the above code, the initial observation table is compiled after setting up the basic alphabet and their respective membership queries. After a failed conjecture, the algorithm requests certain membership query to the teaching algorithm and updates its table accordingly. The while loop ensures the observation table is closed and consistent, checking that it has not been corrupted at any point by additional queries to the teaching algorithm.

Chapter 4

Experiment

4.1 Amazon Mechanical Turk

4.1.1 Objective

The first experiment aims at measuring any differences in human cognitive abilities to acquire through example the definition of conservative quantifiers versus the definition of non-conservative quantifiers. Because no common existing language envelope the defintion of non-conservative quantifiers into a single word, humans are expected to have a bias for more easily acquiring the definition of conservative quantifiers.

4.1.2 Background

The Amazon Mechanical Turk community enables computer programmers, known as requesters, to access to a large-scale group of workers for experiments involving human intelligence, which cannot otherwise be achieved. These requesters upload human intelligence tasks, called HITs, which are perused by workers until choosing to complete the HIT, often for a monetary reward. Requesters may filter the results by approving appropriate HITs or rejecting HITs with unsatisfactory or incomplete responses. The responses by the requesters affect the workers' reputation, which may serve as a qualification for future requesters.

4.1.3 Setup

For the purposes of simplicity, first order Aristotelian quantifiers were chosen as the conservative quantifiers whose definitions were to be identified, and the analagous non-conservative quantifiers were selected. The non-sensical words "zag", "noto", "wim", and "geno" were created and each randomly assigned the follow definitions:

- **zag**: None of the

  Example: Zag of the circles are green $\Leftrightarrow$ none of the circles are green.

- **wim**: One of the

  Example: Wim of the circles are green $\Leftrightarrow$ exactly one of the circles is green.

- **noto**: None of the set excluding

  Example: Noto of the circles are green $\Leftrightarrow$ none of the objects which are not circles are green.

- **geno**: One of the set excluding

  Example: Geno of the circles are green $\Leftrightarrow$ exactly one of the objects which are not circles is green.

Each worker initiating a HIT is greeted by the following message:

> In this experiment, various words will have predetermined definitions, which will be demonstrated in a series of examples, followed by a separate image in which YOU should decide if the word is applied CORRECTLY. Then, if appropriate, enter what you think the definition of the word means.

The worker is then shown four examples of the correct application of a quantifier, followed by another image, at which point the worker is prompted if the application of the quantifier to that image is correct, and what the worker believes the quantifier means. The process is then repeated for each quantifier. A list of images appear in the appendix.

4.2 Artificial Decider

The first implementation of the artifical learner involved no oracle and no informant. Instead the learner would only be given training examples to see whether or not an efficient learner could be construcuted. After seeing several training examples, the algorithm would then decide on a specific case where a certain property applied to a subset of the objects to see whether the quantifier applied or not.

4.3 Artificial Intelligence Learner

4.3.1 Objective

The final implementation of an artificial learner attempts to use the algorithm developed by Angluin to acquire the definition of elementary quantifiers. Although Angluin and later Clark both cite the same algorithm, and prove theoretical results about the algorithm, neither party reportedly implemented it.

4.3.2 Background

A theorem of Angluin states that the total runtime of the learner is bounded by a polynomial in n and m, where n is the number of states of the minimum deterministic finite automata accepting the regular language and m is the upper bound of the length of any counterexample provided by the teacher [1]. However, the problem of identifying the correct determinative finite automata with the minimal number of states is an NP-hard problem.

4.3.3 Setup

The program is initially constructed with the unknown quantifier being "all", whose state machine [3] appears in Figure 4-1: Because of the simplicity of dictionaries in the Python programming language, this project was coded in Python 2.7 with Intel Core Dual 2.53 GHz processors and 4.00 GB RAM.

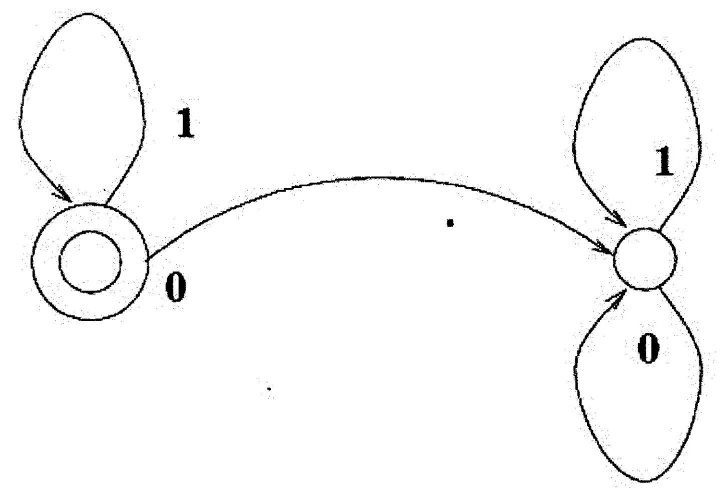

Figure 4-1: Finite state automata of the "all" quantifier.

Chapter 5

Results and Discussion

5.1 Amazon Mechanical Turk

In total 85 participants selected to attempt the task. However, upon first review of the resulting statistics, a staggeringly high 41% of results were rejected, with responses deemed unsatisfactory. Due to the easily exploited nature of Amazon Mechanical Turk, many candidates would rather submit quick responses in hopes of obtaining a reward instead of looking for accuracy. Therefore, only 50 approved respondents completed the task. However, once unsatisfactory results are filtered, the data reveals an amazing discrepancy. The results are summarized in Table 5.1.

Table 5.1: Quantifier success rates in percentages

	zag	wim	noto	geno
Approved, correct application	90%	90%	84%	32%
Total, correct application	74%	81%	75%	36%
Approved, correct definition	60%	64%	4%	6%

Recall the following definitions:

- **zag**: None of the

  Example: Zag of the circles are green $\Leftrightarrow$ none of the circles are green.

- **wim**: One of the

  Example: Wim of the circles are green $\Leftrightarrow$ exactly one of the circles is green.

- **noto**: None of the set excluding

  Example: Noto of the circles are green $\Leftrightarrow$ none of the objects which are not circles are green.

- **geno**: One of the set excluding

  Example: Geno of the circles are green $\Leftrightarrow$ exactly one of the objects which are not circles is green.

In total, 60% of the participants correctly determined the definition of the "zag" quantifier to being equivalent to "none". Moreover, 90% of the participants correctly determined whether or not the "zag" quantifier applied to the example diagram. Once again 90% of the participants correctly determined whether or not the "wim" quantifier applied to the example diagram. Similarly, 64% of the approved respondents correctly specified the definition of the "wim" quantifier, higher than the rate of respondents correctly defining the "zag" quantifier. The success rates for the non-conservative quantifiers were much lower. While 84% of the approved participants correctedly determined if the "noto" quantifier applied to the diagram, only two respondents deduced the correct definition. All other respondents attempted to note either the number of colored squares or the spatial configuration of the squares, instead of looking at the objects which were not squares, despite all diagrams being recycled from the "zag" example. Only 32% of the approved respondents correctly determined if the "geno" quantifier applied to the test diagram, a statistically significant amount, as demonstrated in the calulations below:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{S_{X_1}^2 + S_{X_2}^2}{2n-2}}}$$

$$\begin{aligned} &= \frac{.9 - .32}{\sqrt{\frac{0.09^2+0.2176^2}{98}}} \\ &>> 3 \end{aligned}$$

The t value implies significance ever at the 99.5% confidence level. Because respondents did worse than they would have with a random guess, either the definition of "geno" or the training examples were misleadingly difficult. However, three respondents correctly determined the definition of "geno", refuting the impossibility of acquiring such a quantifier.

5.2 Artificial Decider

The initial philosophy of the decider was to deem a case of the quantifier false unless sufficient evidence proves otherwise.

5.2.1 Training Examples with Percentages

The algorithm would observe training examples in which case the algorithm applied, and keep track of the percentages for the examples. Therefore, the algorithm would ask for examples in which the quantifier was true, and ask for both the total number of objects, as well as the number of objects for which the property applied. The algorithm keeps track of the lower and upper bounds for each case. For example, if the unknown quantifier is "at least half of", possible training examples are 3 of 5, 16 of 19 and 8 of 8. If all three training examples were used, the algorithm would assume the quantifier applies if property applies to anywhere between 60% and 100% of the objects. However, this method would never be able to encapsulate the idea of "some", which notes any percentage greater than 0%. In a sequence of training examples where n is the largest number of objects in any training example, the algorithm will think "some" is false if the property applies to 1 object out of $n + 1$ objects.

5.2.2 Training Examples with Flat Numbers

Hence, the algorithm was modified so that it would also keep track of the flat numbers of objects. Hence, if the training examples are 3 of 5, 16 of 19 and 8 of 8 the algorithm would assume the quantifier applies if property applies to anywhere between 60% and 100% of the objects or if the property applies to anywhere between 3 and 8 of the objects. This addition solves the problem of "some". In fact, the modification allows any Aristotelian first order, cardinal first order, or proportional quantifier to be learned, with the proper training examples. However, because only positive training examples are used, there is no way for parity quantifiers such as "an even number of" to be learned. Even with the addition of negative training examples, parity quantifiers cannot be learned under this type of decider.

5.3 Artificial Intelligence Learner

5.3.1 Constructing the teacher

Although the notation was initially intractable, the pseudo-code proved less resistant. After the first successful compilation of the learner, the teacher algorithm was constructed. For simplicity purposes, the minimally adequate teacher algorithm was instructed to check conjectures up to twelve characters long. Therefore, counterexamples would be given in dictionary order. That is, the counterexample "000011011010" would be given before the counterexample "111111000111". Additionally, the teacher algorithm would accept any string with all digits comprised of 1's and reject strings otherwise. Initially the interaction between the mapping function T and the current state Q was incorrect, resulting in several key errors in the T dictionary. Consequently, the relationship was changed so that at every step in $M(S, E, T)$, the transition function would search for s such that $row(s \cdot a) = row(s)$ and record s as the current state, instead of $row(s)$.

Table 5.2: Output for $M(S, E, T)$ for the "all" quantifier

S	[‘’, ‘0’]
E	[‘’]
T	(‘01’, ‘’): ‘0’, (‘00’, ‘’): ‘0’, (‘0’, ‘’): ‘0’, (‘’, ‘’): ‘1’, (‘1’, ‘’): ‘1’

5.3.2 Initial results

The resulting configuration for the "all" quantifier appears in Table 5.2. Compare this table to the finite state automata in Figure 4-1 which represents the quantifier "all". The initial state represented by λ is also the accepting state, and transitions to itself with the occurrence of input ‘1’. The other state, represented by ‘0’ is not an accepting state, and transitions to itself regardless of input, and can be reached from the initial state by input ‘0’ at any time. In addition, the rows ‘0’, ‘1’, ‘00’ and ‘01’ are represented because of the two states and the alphabet {‘0′, ‘1′}.

5.3.3 Other two state quantifiers

With the success of the Angluin algorithm in acquiring the definition of the "all" quantifier and determining the corresponding finite state automata with the minimal number of states, the algorithm was repeated with other quantifiers to see the resulting outputs. In each case, both the algorithms which gave the result of membership queries and the teaching algorithms were modified correspondingly. The quantifier "none" serves as almost an exact replica of the "all" quantifier in its formulation of the state machine, as demonstrated by the comparison of Figure 5-1 to Figure 4-1. Unsurprisingly, the output $M(S, E, T)$ for "none" is exceedingly similar, shown in Table 5.3. The initial state represented by λ is also the accepting state, and transitions to itself with the occurrence of input ‘0’, instead of ‘1’ as in the "all" case. The other state, represented by ‘1’ is not an accepting state, and transitions to itself regardless of input, and can be reached from the initial state by input ‘1’ at any time. In addition, the rows ‘0’, ‘1’, ‘10’ and ‘11’ are represented because of the two states

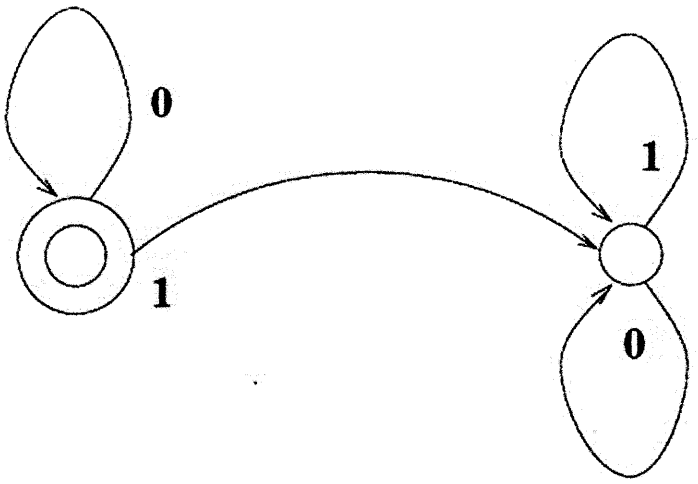

Figure 5-1: Finite state automata of the "none" quantifier.

Table 5.3: Output for $M(S, E, T)$ for "none" quantifier

S	['', '1']
E	['']
T	('11', ''): '0', ('0', ''): '1', ('', ''): '1', ('10', ''): '0', ('1', ''): '0'

and the alphabet {'0','1'}.

Additionally, the parity quantifier, "an even number of", which could not have been implemented under the previous model of the artificial decider, also has two states in its minimal representation as a finite state automata. Fortunately, the Angluin algorithm was able to produce the correct finite state automata with the minimal number of states. The difference between the resulting $M(S, E, T)$ for this quantifier and previous quantifiers is the number of entries $s \in (S \cup S \cdot A)$ for which $T(s) = 1$, which reflects the possibility that when the state machine leaves the accept state in this case, it can still return to the accept case.

Table 5.4: Output for $M(S, E, T)$ for "even" quantifier

S	['', '1']
E	['']
T	('11', ''): '1', ('0', ''): '1', ('', ''): '1', ('10', ''): '0', ('1', ''): '0'

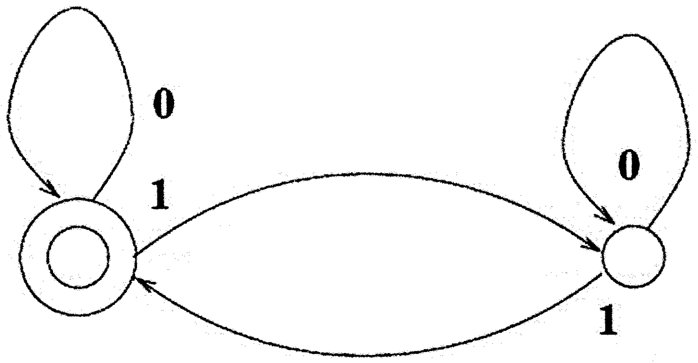

Figure 5-2: Finite state automata of the "even" quantifier.

5.3.4 Larger number of states

How well does the Angluin algorithm handle quantifiers which need more than two states in its minimal representation as a finite state machine? An elementary quantifier with more than two states is "a multiple of 3 of", as in *a multiple of 3 of the circles are green.* For the remainder of the paper, this quantifier shall be called *three.* The associated finite state machine is displayed in Figure 5-3 and the corresponding output $M(S, E, T)$ is shown in Table 5.5. The output is vastly different from the previous outputs, and with good reason. Obviously, the number of states has increased to three, as desired. However, E is finally nonempty, which reveals the crux of the algorithm.

When $M(S, E, T,)$ evaluates the string '111', it starts in the state $row(\lambda)$, which is an accept state. Then by definition,

$$\delta(\lambda, 1) = row(1) = 0,$$

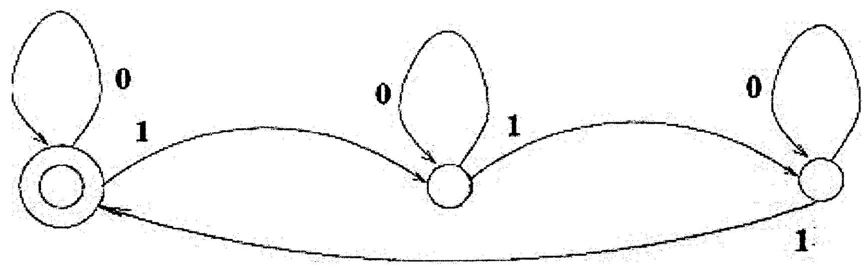

Figure 5-3: Finite state automata of the “three” quantifier.

Table 5.5: Output for $M(S, E, T)$ for “three” quantifier

S	[‘’, ‘1’, ‘11’]
E	[‘’, ‘1’]
T	(‘11’, ‘’): ‘0’, (‘110’, ‘’): ‘0’, (‘11’, ‘1’): ‘1’, (‘0’, ‘’): ‘1’, (‘’, ‘1’): ‘0’, (‘111’, ‘’): ‘1’, (‘’, ‘’): ‘1’, (‘110’, ‘1’): ‘1’, (‘1’, ‘1’): ‘0’, (‘10’, ‘’): ‘0’, (‘111’, ‘1’): ‘0’, (‘0’, ‘1’): ‘0’, (‘10’, ‘1’): ‘0’, (‘1’, ‘’): ‘0’

which is a reject state. In the next transition,

$$\delta(1, 1) = row(11) = 0,$$

which is again a reject state. Unfortunately, $M(S, E, T)$ has no way method of correctly assigning the reject state. By design, $M(S, E, T)$ selects the first row which is a reject state. Therefore, the algorithm selects $row(1)$. But then

$$\delta(11, 1) = row(11) = 0,$$

which results in a reject state. That is, $M(S, E, T)$ rejects ‘111’ while the teacher accepts, which causes a membership query of all prefixes and requests another conjecture from the algorithm. However, all prefixes of ‘111’ are already contained in S and correctly defined in T, so neither S nor T will change, which would in the exact same conjecture, causing the algorithm to loop indefinitely. Fortunately, the construction of E serves as a state memory, so that for multiple reject states, some column of E

will differentiate the rows. Therefore, after the first failure of the ‘111’ input, the learner realizes that the table is not consistent, because the current rows of ‘1’ and ‘11’ are the same, but behave differently upon an additional ‘1’. Hence, ‘1’ is added to E and the membership queries are run. This time, the two rows are different, and the learner recognizes the correct row to pass, and the conjecture is correct.

Chapter 6

Summary

6.1 Cognitive Bias

The results from the Amazon Mechanical Turk experiment reveals a cognitive bias for humans to search for conservative quantifiers, as expected, since no non-conservative quantifier is expressed as a single word in any language. However, because Amazon Mechanical Turk does not grant the ability for the same respondent to complete multiple tasks and be timed for each task, it remains to be seen whether or not the verification process for humans is also biased toward conservative quantifiers. Perhaps if the definitions of a conservative quantifier and its analogous non-conservative quantifier were given, humans might also show the ability to verify the conservative quantifier faster than the non-conservative quantifier.

6.2 Artificial Intelligence

Although the artificial decider was able to acquire the definitions of several quantifiers using the combination of positive training examples of both percentages and flat numbers, no amount of training examples would permit the artificial decider to learn parity quantifiers. Moreover, if random training examples were input into the artificial decider, than the accuracy of the decider would be no better than an exercise in mathematical probability: whether or not the test case falls within the range of the previous training examples.

On the other hand, the artificial intelligence learner outlined by Angluin seems perfectly capable of returning the correct finite state automata of 'all' with the minimum number of states. Moreover, it successfully acquired the correct representation of "none" and "an even number of", demonstrating that the Angluin algorithm can acquire both Aristotelian first order quantifiers and parity quantifiers, the latter of which the artificial decider is unable to acquire. Because cardinal first order quantifiers can still be represented by regular languages, the Angluin algorithm can acquire their definitions through a minimally adequate teacher as well. However, proportional quantifiers cannot be expressed by regular languages, so they cannot be captured by the Angluin algorithm. Moreover, the nested for loops would likely slow computation for more complicated quantifiers with far greater than number of states. In fact, the process of confirming consistency relies on iterating twice through the prefixes and once through the suffixes, the sizes of which can be on the order of the number of states. Similarly, the number of incorrect conjectures and subsequent membership queries can also be on the order of the number of states. Therefore, the resulting runtime for the algorithm applied to complex quantifiers can easily be asymptotically quartic on the number of states in the minimal representation of the quantifier. Lastly, the presence of a minimally adequate teacher may be somewhat optimistic, as with the existence of an oracle, learners naturally become easier.

6.3 Alternative Forms of Identification

To justify the selection of identifiable in the limit as the standard for identification, alternative standards must be analyzed. *Finite identification* is the most common identification problem, especially in automata theory. Under this standard, the learner will stop accepting new information if it thinks it has received sufficient information in order to decide upon a language. However, the decided language cannot surely be correct unless all other languages have been disproved, which means that a sequence of information can be constructed to disprove the existence of a completely accurate learner under finite identification.

Fixed-time identification forces the learner to decide upon a language after a fixed amount of finite time, independent of the unknown language and text. Similar to the previous identification standard, some text can always be constructed to disprove the existence of a completely accurate learner under fixed-time identification.

The basic idea in the counterproofs previously mentioned is to create a sequence with enough useless information to stall the learner until the number of same continuous guesses or the time is exceeded. Hence, the idea of redundant information in text is powerful enough to render enough the most carefully constructed learner either incorrect, as in either of the later two identification models, or highly inefficient, as such under the identifiable in the limit model. Thus, perhaps by restricting the text that can be input into a learner, better results can be achieved.

Appendix A

Figures: Zag, Noto

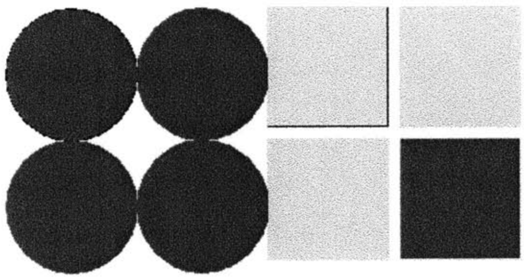

Figure A-1: Zag of the circles are green / Noto of the squares are green: 1

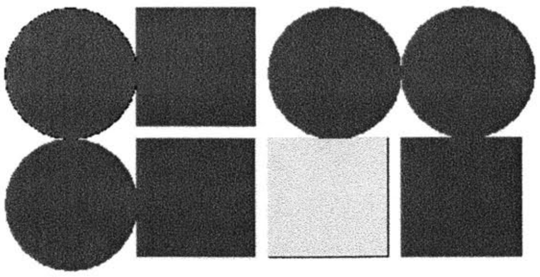

Figure A-2: Zag of the circles are green / Noto of the squares are green: 2

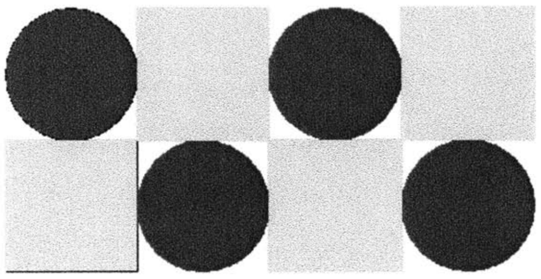

Figure A-3: Zag of the circles are green / Noto of the squares are green: 3

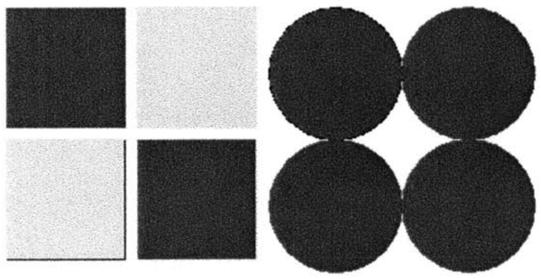

Figure A-4: Zag of the circles are green / Noto of the squares are green: 4

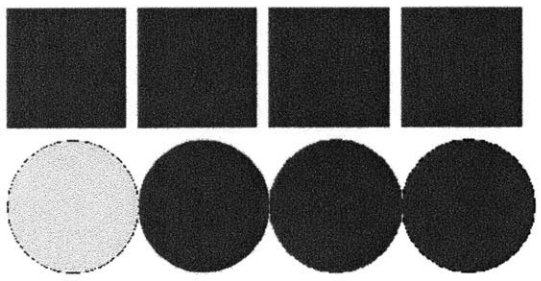

Figure A-5: Zag of the circles are green / Noto of the squares are green: False test case

Appendix B

Figures: Wim, Geno

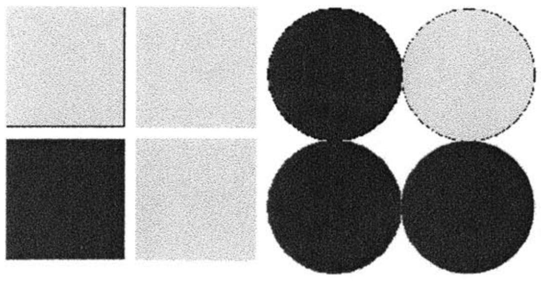

Figure B-1: Wim of the circles are green / Geno of the squares are green: 1

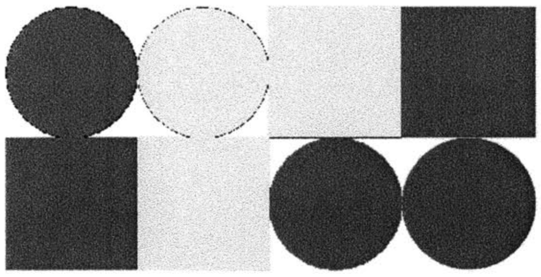

Figure B-2: Wim of the circles are green / Geno of the squares are green: 2

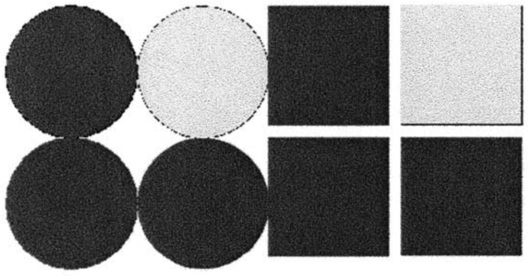

Figure B-3: Wim of the circles are green / Geno of the squares are green: 3

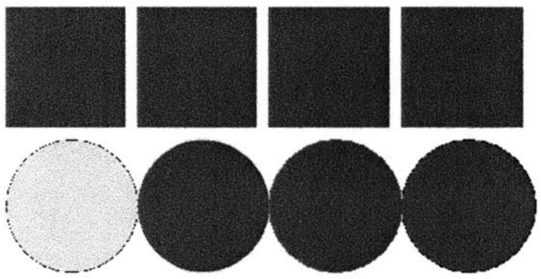

Figure B-4: Wim of the circles are green / Geno of the squares are green: 4

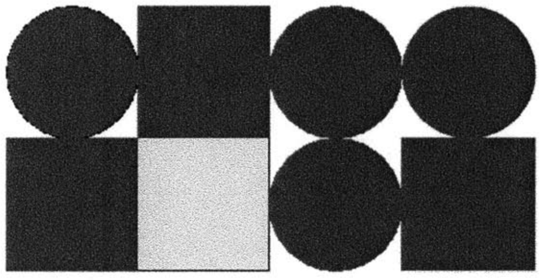

Figure B-5: Wim of the circles are green / Geno of the squares are green: False test case

Bibliography

[1] Angluin, D. "Learning Regular Sets from Queries and Counterexamples." Information and Computation 75, 87-106 (1987)

[2] Berwick, R. & Gilbert, S. "Learning Quantifiers from Examples: A Linguistic and Bayesian Approach." MIT

[3] Clark, R. "Learning First Order Quantifer Denotations An Essay in Semantic Learnability." IRCS Technical Report Series. 1996.

[4] Gierasimczuk, N. (2007). "The Problem of Learning the Semantics of Quantifiers." LNAI 4363, 117-126

[5] Gold, E. (1967). "Language Identification in the Limit." Information and Control 10, 447-474.

[6] Hackl, M. "On the grammar and processing of proportional quantifiers: most versus more than half." 3 March 2009.

[7] Hurewitz, F., Papafragou, A., Gleitman, L., & Gelman, R. (2006). "Asymmetries in the Acquisition of Numbers and Quantifiers." Language Learning and Development 2, 77-96.

[8] Szymanik, J. & Zajenkowski, M. (2010). "Comprehension of Simple Quantifiers: Empirical Evaluation of a Computational Model." Cognitive Science 34, 521-532.