PSYCHOCOMPUTATIONAL MODELS OF SUBSET PRINCIPLE COMPLIANCE IN SIMULATED LANGUAGE LEARNING

By

ARTHUR HOSKEY

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2008

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

This manuscript has been read and accepted for the
Graduate Faculty in Computer Science in satisfaction of the
dissertation requirement for the degree of Doctor of Philosophy.

Dr. William Gregory Sakas

April 25, 2008 — Chair of Examining Committee

Date

Dr. Theodore Brown

April 25, 2008 — Executive Officer

Date

Dr. Janet Dean Fodor, Graduate Center

Dr. Virginia Teller, Hunter College

Dr. Damir Ćavar, The University of Zadar

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract:

Psychocomputational Models of Subset Principle Compliance in Simulated Language Learning

By

Arthur Hoskey

Advisor: Dr. William Gregory Sakas

Previous research has proposed that any model of language learning should use the Subset Principle to guide hypothesis selection when the language domain contains at least two languages such that one is a subset of the other (Gold, 1967; Berwick, 1985; Manzini & Wexler, 1987; Wexler & Manzini, 1987). Informally, the Subset Principle states that the learner should select a language that: a) is compatible with the input data, and, b) does not properly contain any other language that is compatible with the input data. This thesis puts forth a comprehensive investigation of psychocomputational models of language learning that abide by the Subset Principle from both an empirical and theoretical perspective. We intend "psychocomputational models" to include computational models that are in line with research in psycholinguistics, developmental psychology and theoretical linguistics (Sakas, 2004). This thesis is divided into three principal areas: 1) an analysis of partial ordering learners which are given a priori knowledge of subset-superset relationships, 2) a comparison of those partial ordering learners and variants of traditional Gold-paradigm total ordering (enumeration) learners, and 3) a preliminary investigation into how the shape of the language domain, in terms of both the partial ordering of

subset-superset relationships and cross-language ambiguity, affects learning performance of learners that abide by the Subset Principle. Results show that the partial ordering learners perform best when Subset Principle constraints and parsing are given equal weight with regards to hypothesis selection. The comparison study shows that the partial ordering learners outperform the total ordering learners. Finally, preliminary results stemming from the investigation of language domain shape indicate that language domains exhibiting greater breadth than depth are more computationally demanding to learn. Although there exists a number of computational studies that attempt to address the problems that are introduced when learning in domains that contain superset languages, this research makes its contribution by modeling the Subset Principle under assumptions that are psychologically realistic in terms of the computational workload required of the learning algorithms under investigation.

Acknowledgements

First off, I would like to thank my advisor, Dr. William Sakas, for helping to guide me through the dissertation process. I sought out Dr. Sakas because he was active in the research community and, more importantly, I knew that he would actually help me to *learn* how to do research. He went above and beyond the call of duty. He has always been generous with both his time and knowledge. His assistance and encouragement have been invaluable to me and I will always be grateful. I would like to thank Dr. Janet Dean Fodor for all of her ideas and suggestions as well as editing throughout the whole process. Thank you to Dr. Virginia Teller for serving on my committee and for hiring me for my first teaching assignment and to Dr. Damir Ćavar for finding the time to serve on my committee from half way around the world. I would also like to thank Dr. Theodore Brown for not giving up on me in the Computer Science program even when things were slower than we both would have liked them to be.

I would like to thank my mother Frances and father Gerard for their support throughout my life. They were always encouraging and they always believed in me. My mother was my biggest supporter when I struggled through school as a child. She stuck by me through thick and thin and never wavered. My father was always a model to me of what a person should strive to be. He was a loving man of high integrity, intelligence, and an extremely hard worker. There was and there still is no one I

wanted to be like more than him. I love him and I miss him dearly. I wish he could have been here to see me finish but I know he would have been very proud.

I would like to thank Dr. Susan Valicenti for her support. I have had many struggles throughout this process and she always managed to find the right thing to say to me to help me keep going. Always caring and supportive, I will always be thankful for the large part that she has played in helping me live life to the fullest.

Many people helped me while growing up and I would like to thank them because I would not be where I am right now if it weren't for them. Many thanks to Jonathan Berent, M.S.W. who was a skilled social worker that helped me to conquer the many anxieties that I experienced while growing up. He was instrumental in helping me transform from a scared boy to a confident man. Thanks to my cousins Jimmy and Kathy as well as my Aunt Dora and my Grandma Rose for their support and companionship. Thank you to my friend Evan Ziegler, as well as his parents, Arthur and Elaine, for letting me hang around their house for all of those years. I would like to send a very special thanks to Michael Kucker. Mr. Kucker taught me as a home school instructor during high school and helped me through the hardest period in my life. I struggled socially and academically and he was there for me when I needed it the most.

I would like to thank Dr. Julia Beltsiou, Amanda Pustam, Evelyn Neunteufel and Sophie Saint-Just from the dissertation workshop group for all of their support and

encouragement. Thanks to Dr. Alissa Schamber for her assistance during my final push to finish my dissertation. Thanks to my fellow Computer Science graduate students who participated in the comprehensive exam study group. You were invaluable in helping me get through that exam. Thanks to Mirella Piccone for her excellent help and support in helping me learn Syntax.

I would like to thank all of my family and friends who have supported me through this process. My sons, Aidan Hoskey and Gareth Yates, who helped me to have fun when I needed it. Gareth was also like a second father to Aidan and was a big help in allowing me to spend time working on my dissertation. Dr. Charles Jeremy Sykes for supporting me and for helping me to move from being a student to a researcher with a voice. Tari Lee Sykes for putting up with me bringing my dissertation to family gatherings and also taking care of Aidan when I needed the time. Thanks to Susan Sykes for her kind words and encouragement. Susan also spent lots of time taking care of Aidan, which allowed me flexibility with my job and schoolwork. My friends Peter Brown, Tracy Brown, Bernadette Smith, Henry Smith, and Michelle Malone who supported, pushed and encouraged me throughout.

Finally, I would like to thank my fiancée Cindy Parker. The whole dissertation process has been a marathon for me to say the least and she has been supportive every step of the way. So many things have happened throughout and she has stayed right there with me. I have been in and out of jobs (currently out), I took a leave of absence, I changed advisors (which meant starting research over after a whole year's

worth of work) and most importantly she gave birth to our son Aidan. She has proof read, helped me with statistics and helped me with research ideas when I got stuck. She has been right there by my side through all of the trials and tribulations. She is the definition of “Superwoman”. I love her.

Table of Contents

Table of Contents x
List Of Tables xiii
List of Figures xv
1 Introduction 1
1.1 Formal Learning Theory in the Gold Paradigm 3
1.1.1 Identification In The Limit 3
1.1.2 Telltale Sets 6
1.2 Computational Models of Syntactic Parameter Setting 9
1.2.1 Triggers 10
1.2.1.1 Triggering Learning Algorithm 10
1.2.1.2 The Deterministic Structural Triggers Learner 12
1.2.2 Stochastic Methods 18
1.2.2.2 Genetic Algorithm Learner 19
1.2.2.2 Naïve Parameter Learner 21
1.2.2.3 Guessing STL 24
1.2.2.4 Probabilistic Components of the TLA 24
1.2.3 Feasibility 25
1.2.3.1 Learning From Triggers 26
1.2.3.2 Ambiguity and the Computational Feasibility of Syntax Acquisition 28
1.2.3.3 Modeling the Effect of Cross-Language Ambiguity on Human Syntax Acquisition 32
1.2.3.4 Model Comparisons 35
1.3 The Subset Principle 37
1.3.1 Subset Principle Background 38
1.3.1.1 Berwick 38
1.3.1.2 Manzini and Wexler 39
1.3.2 Arguments Against the Subset Principle 42
1.3.2.1 Becker 42
1.3.2.2 MacLaughlin 43
1.3.3 Subset Principle Compliance 44
1.3.3.1 The Shifting Problem 44
1.3.2.2 The Simple Defaults Model 46
1.3.3.3 The Ordered Defaults Model 47
1.3.3.4 Effects of Incremental Learning 48
1.3.3.5 Subset-free Triggers 49
1.4 The Simulation Platform 50
1.4.1 CoLAG Domain 50
1.4.2 Simulation Program 51
2 Partial Ordering Learners 57
2.1 Justification For Adding Memory For Disconfirmed Grammars 57
2.2 General SP Lattice Learner 58
2.3 SP Lattice Learner Variants 61
2.3.1 Parallel Parsing Vs. Serial Parsing 62

2.3.2 Decoding 63
2.3.3 Decode Learner 63
2.3.4 Decode Favor Unmarked Learner 65
2.3.5 Integrated Learner 67
2.3.6 Flashlight 69
2.3.7 Largest Language Optimal Learner 71
2.3.8 Retrench Learner 75
2.4 Results and Discussion 79
2.4.1 LL Optimal Best For Sentences but Not Parses 80
2.4.2 Integrated Learner Performs Best 80
2.4.3 Higher Parsing Priority Hinders The Decode Learner 82
2.4.4 CoLAG Is Unlearnable For The Decode Favor Unmarked Learner 84
2.4.5 Impact of the Flashlight 88
2.4.6 Retrench Learner Is Good But CoLAG Is Unlearnable 91
2.4.7 The Parser and SP Work Best in Tandem 93
3 Comparison of Partial and Total Ordering Learners 96
3.1 Why Use A Partial Ordering? 97
3.2 Total Ordering Learners 97
3.3 Gold's Total Ordering Learner 98
3.4 Memoryless Total Ordering Learner 100
3.5 Constrained Memoryless Total Ordering Learner 102
3.6 Discussion and Results 104
3.6.1 Total Ordering Learner Inefficient In Terms of Parses 104
3.6.2 Effects of Removing Total Ordering Input Sentence Memory Store 110
3.6.3 Constraining Total Ordering Learner Affects Performance 111
3.6.4 Partial Ordering More Efficient Than Total Ordering 112
4 Effects of Language Domain Shape on Learning 115
4.1 Why Examine Language Domain Shape? 116
4.2 Tall Vs. Wide Lattices 116
4.3 Domain Ambiguity Within The Language Domains 117
4.4 Subset Language Domains 118
4.4.1 Description of Subset Language Domains 118
4.4.1.1 Language Domain Shape - 5-45-45-5 118
4.4.1.2 Language Domain Shape - Skewed 120
4.4.1.3 Language Domain Shape – 10 x 10 121
4.4.1.4 Language Domain Shape – 50 x 2 121
4.4.1.5 Language Domain Shape – 25 x 4 123
4.4.1.6 Language Domain Shape – 4 x 25 123
4.4.1.7 Language Domain Shape – 2 x 50 124
4.4.2 Performance Across Subset Language Domains 125
4.4.2.1 SP Lattice Learner Performance Across Language Domains 125
4.4.2.2 Retrench Learner Results 127
4.4.2.3 Total Ordering Learner Results 131
4.4.2.4 Total Ordering Constrained Memoryless Learner Results 133
4.4.2.5 SP Lattice Flashlight Learner Results 134
4.4.2.6 Discussion 135

4.5 Properly Intersecting Language Domains 137
4.5.1 Description of Properly Intersecting Language Domains 139
4.5.2 Learner Performance On Properly Intersecting Language Domains 140
4.6 Discussion 143
5 Conclusions, Implications and Future Research 147
5.1 Summary of Findings 147
5.2 Future Research 150
Appendix A – Miscellaneous Procedure Descriptions 153
Appendix B – Pseudocode Guide 156
References 157

List Of Tables

Table 1: Lattice learner results on CoLAG domain. 79

Table 2: Pearson r correlations for Lattice and Flashlight learners. 89

Table 3: Retrench learner performance on selected languages from the CoLAG domain. 93

Table 4: Comparision of Total Ordering and Lattice learners 104

Table 5: SP Lattice learner results across language domains. 125

Table 6: Retrench learner results across language domains. 127

Table 7: Retrench learner results with languages grouped by row for the 5-45-45-5 language domain. 128

Table 8: Total Order learner results across language domains. 131

Table 9: Total Ordering Constrained Memoryless learner results across language domains. 133

Table 10: SP Lattice Flashlight learner results across language domains. 134

Table 11: Relative performance of learners on language domains in terms of the number of parses (fastest (1) to slowest (7)). 135

Table 12: Average number of subsets per language by language domain. 136

Table 13: Average number of parses for languages in either the top or bottom row of the 2x50 language domain. 137

Table 14: SP Lattice learner results across properly intersecting language domains. 140

Table 15: Retrench learner results across properly intersecting language domains. 141

Table 16: Total Ordering learner results across properly intersecting language domains. 141

Table 17: Total Ordering Constrained Memoryless learner results across properly intersecting language domains. 142

Table 18: SP Lattice Flashlight learner results across properly intersecting language domains. 142

List of Figures

Figure 1: Shifting problem. 45

Figure 2: Example language domain learnable with ODM but not SDM. 47

Figure 3: Learner base class. 52

Figure 4: Main processing loop. 53

Figure 5: Global variables. 54

Figure 6: An example learner class. 55

Figure 7: Setting the first hypothesis for the example learner algorithm. 55

Figure 8: Resetting the example learner for a new trial. 55

Figure 9: Example learner algorithm. 56

Figure 10: Remove language 1 from SL and the lattice. 59

Figure 11: SPLatticeLearner class. 60

Figure 12: Lattice learner algorithm. 61

Figure 13: SPLatticeDecodeLearner class. 64

Figure 14: SP Lattice Decode learner algorithm. 64

Figure 15: SPLatticeDecodeFavorUnmarkedLearner class. 66

Figure 16: SP Lattice Decode Favor Unmarked learner algorithm. 66

Figure 17: SPLatticeIntegratedLearner class. 67

Figure 18: Integrated learner algorithm. 68

Figure 19: SPLatticeFlashlightLearner class. 70

Figure 20: SP Lattice Flashlight learner algorithm. 71

Figure 21: SPLatticeLLOptimalLearner class. 73

Figure 22: LL Optimal learner algorithm. 74

Figure 23: RetrenchLearner class. 76

Figure 24: Retrench learner algorithm. 77

Figure 25: Retrench procedure..... 78

Figure 26: TotalOrderingLearner class. 99

Figure 27: Gold's Total Ordering learner algorithm..... 100

Figure 28: TotalOrderingMemorylessLearner class. 101

Figure 29: Memoryless Total Ordering learner algorithm. 101

Figure 30: TotalOrderingConstrainedMemorylessLearner class. 102

Figure 31: Constrained Memoryless Total Ordering learner algorithm. 103

Figure 32: Language domain with ambiguous input sentences. 108

Figure 33: Language Domain Shape 5-45-45-5 Fully Connected 119

Figure 34: Language Domain Shape Skewed Fully Connected 120

Figure 35: Language Domain Shape 10 x 10 Fully Connected 121

Figure 36: Language Domain Shape 50 x 2 Fully Connected 122

Figure 37: Language Domain Shape 25 x 4 Fully Connected 123

Figure 38: Language Domain Shape 4 x 25 Fully Connected 124

Figure 39: Language Domain Shape 2 x 50 Fully Connected 124

Figure 40: Target language that is hard for the Total Ordering learner to acquire... 132

Figure 41: Language domain with high domain ambiguity. 138

Figure 42: Language domain with low domain ambiguity. 139

Figure 43: Language shape that is hard for the Retrench learner. 144

Figure 44: Language shape that is easy for the Retrench learner. 145

1 Introduction

The main goal of this research is to investigate psychocomputational models of learning that abide by the Subset Principle (SP). First proposed by Gold (1967), though the term is generally attributed to Berwick (1985), Manzini and Wexler (1987) and Wexler and Manzini (1987). We will use Fodor and Sakas (2005) definition of SP. Note that for Fodor & Sakas a *smallest language* means a language in a learning domain that does not properly contain any other language in the domain (i.e., a smallest language does not contain a subset language).

> When the learning mechanism's current language is incompatible with a new input sentence *i*, the learning mechanism should hypothesize a UG-compatible language which is a smallest superset of i *and all prior input sentences retained in its memory*, excluding any language *recorded in memory as having been disconfirmed by prior input*.
>
> Fodor and Sakas (2005)

The Subset Principle will be used to help the learners navigate the search space. SP is a rule that learners employ in order to avoid getting stuck in a state from which they cannot escape. The need for SP arises when a language domain contains two languages such that one is a subset of the other. Most, if not all, computational models of human language learning at the present time do not abide by the Subset Principle (see discussion in Gibson and Wexler, 1994; Sakas and Fodor, 2001). There is an abundance of research that shows that it is difficult for a learner to navigate the

search space of possible languages to find the target language, even with supersets of the target removed (Clark, 1992; Gibson and Wexler, 1994; Briscoe, 2000; Sakas and Fodor, 2001; Yang, 2002). Clearly, adding supersets to the language domain makes the problem even harder. For learners that do not change their hypothesis unless the current input contradicts it, hypothesizing a superset of the target would cause the learner to fail. The current research is to develop learners that can successfully learn in an environment containing supersets of the target language.[1] The learners being developed are constrained such that they are allowed only two parses per input sentence. The motivation for this is to develop learners that embody psycholinguistically viable constraints. The models under investigation will be endowed with memory for past grammars (Fodor and Sakas, 2005). A comparison study was also done which compares our partial ordering learners against traditional total ordering learners. In addition, we investigate how the shape of the language domain affects learner performance.

[1] Throughout I will freely interchange the terms *language* and *grammar*. Based on context I intend either the sentences generated by a grammar, or the grammar itself.

1.1 Formal Learning Theory in the Gold Paradigm

The Gold paradigm was designed to enable the investigation of language learning (Gold, 1967). Gold's paradigm has had a great influence on research in the area of language learnability and computational linguistics. The main result of his research was a proof that all the classes of languages in the Chomsky hierarchy except the class of finite languages are not learnable from a positive presentation of data. Most developmental psychologists believe that children learn language by getting only positive instances of data so this is a very important result. However, further research (Angluin, 1980) has proved that there are other classes of languages that are learnable from positive data. This first section will describe "identification in the limit" which is Gold's definition of learnability. The second section will describe "telltale" sets. Telltale sets are used to identify a target language that is generating the positive presentation of data; if all languages in a class of languages contain a telltale set, then that class is learnable from positive data.

1.1.1 Identification In The Limit

Learners in the Gold paradigm receive infinite sets of input strings and make a hypothesis about the target language after each individual input. A language is considered learnable in the Gold paradigm if it can be "identified in the limit". Identification in the limit means that after a finite amount of time the learner will

always guess the same language and that language will be the target language. Identification in the limit is a realistic definition of learnability because presumably children do not in fact know for sure when, during the course of learning, they have hypothesized the correct (target) language.

In the Gold paradigm classes are considered learnable with respect to a given information presentation (text or informant). An information presentation method is the way in which a learner receives training data. A text is defined as an infinite set of strings taken from the target language and *only* from the target language. No other strings can appear in a text. All strings of the target language are guaranteed to appear at least once in a text. An informant is defined as an infinite set of pairs of data. Each pair consists of a string and a binary value saying if that string is a member of the target language. Gold breaks down the text and informant types further into subtypes and the results are all the same with one exception (Gold, 1967).[2] The text and informant methods of information presentation can be compared to the ways a child may receive data. If the child is only being given strings of the target language then she is being exposed to a text. If the child is being given corrective information about strings not in the target language, in addition to strings from the target language then she is being exposed to an informant.

Gold defined a guessing rule called identification by enumeration that the learner should use to hypothesize languages. Identification by enumeration proceeds as follows: (1) Enumerate the class of languages in any way such that they all appear at

[2] Anomalous text.

least once. (2) After each input sentence guess the unknown language to be the first language of the enumeration that agrees with the information received so far. It is important to point out that there are enumerations that will not work with a text information presentation for certain classes of languages. For instance, a class of languages that contains subset/superset languages will not be learnable with an enumeration that puts the supersets first. The learner will search the enumeration for the first language that is consistent with the input data so far and that language could be a superset of the target. Since the Gold learner is error-driven it will not change its current hypothesis unless it is necessary. If the learner is hypothesizing a superset of the target then it will never be necessary to change the hypothesis because all sentences in the subset language are also members of the superset language. If the information presentation were changed from a text to an informant though it would be possible to learn from the problematic enumeration because the informant contains negative information that allows it to move from the superset hypothesis.

Gold proved that any class of languages that contains all the finite languages over a vocabulary plus one infinite language that contains those finite languages is not learnable from a text presentation. This means that all the classes in the Chomsky hierarchy except the class of finite languages are not learnable from text alone. Gold believes that the ramifications of this with regards to child learning would be that either (1) The class of possible natural languages is much smaller than expected or (2) the learner receives negative information in a way we do not recognize or (3) there

are constraints on the way positive data are presented to the learner such as the order of presentation (Gold, 1967, p 453).

Finite classes of languages have been shown to be learnable using identification by enumeration under the criterion of identification in the limit (Bertolo, 2001). The enumeration of grammars must have the property that if k>j then either $L(G_k) = L(G_j)$ or there is at least one sentence in $L(G_k)$ that is not in $L(G_j)$. This property of the enumeration is important because it will force the learner to eventually hypothesize the target language. Information presentation by text guarantees that every sentence will show up at least once so we are guaranteed that the sentences the learner needs to move it through the enumeration will appear. This property also guarantees that all subset languages will appear before their respective superset languages so the learner cannot hypothesize a superset of the target. By definition, the number of languages definable within the principles and parameters framework (Chomsky, 1981; Chomsky, 1986) is finite so they are in fact learnable in the limit using identification by enumeration (assuming the enumeration has the property just described above). Depending on the number of parameters, though, it may take an intractable amount of time to converge on the target language. Learning in the principles and parameters framework is theoretically possible but not necessarily feasible.

1.1.2 Telltale Sets

Research has been done which shows that there are certain conditions that when met allow a learner to learn nonempty recursive formal languages from positive data (i.e., text presentation)(Angluin, 1980). Angluin has done research that considers inductive inference of formal languages from positive data in the Gold paradigm (Angluin, 1980). Gold proved no class of languages in the Chomsky hierarchy is learnable from only positive data. Angluin describes classes of recursive languages that are in fact learnable from positive data, however they do not fit neatly as subclasses in the Chomsky hierarchy. The main characteristic of these classes of languages is that every language has a "telltale" set. A telltale set is a finite subset T of a language L such that no other language of the class that contains T is a proper subset of L. If all of the sentences of a telltale set T of L appear in the input stream then the learner can safely hypothesize L and be guaranteed of not commiting a superset error. The existence of telltale sets for each language guarantees that a given class of recursive languages can be learned from positive data.[3]

Sakas and Fodor make use of the idea of a telltale set with their subset-free trigger (Fodor and Sakas, 2005). A subset-free trigger is simply a telltale set with only one item. In the Fodor & Sakas paradigm, there is no enumeration, so subset-free triggers serve as a mechanism to avoid superset hypotheses. In order for a domain of languages to be learnable using an incremental learner[4], without an enumeration, all

[3] More specifically, an indexed family of nonempty recursive languages is inferable from positive data if and only if each language of the given class has a telltale set. An indexed family of nonempty recursive languages is an infinite sequence of nonempty languages such that there is an effective procedure to compute the membership function for each of the languages of the sequence (Angluin, 1980, p 119-121).

[4] Incremental learning means no memory for past inputs or past grammar hypotheses.

of the languages in the domain must have a subset-free trigger. If they don't then the learner runs the risk of chronic undergeneralization if the target language is indeed a superset and doesn't contain a subset-free trigger. This would happen because no input string would exist that would force (trigger) the learner to hypothesize the superset.

1.2 Computational Models of Syntactic Parameter Setting

Syntactic parameter setting models are based on concepts taken from the principles and parameters framework (Chomsky, 1981). The principles and parameters framework is made up of two main components: Universal Grammar (UG) and Parameters. The UG component consists of the principles that are common to all grammars (Chomsky, 1957). An example of a universal principle would be that every sentence must have a subject. The parameters component is made of all the grammatical features that can vary. An example of this would be the null subject parameter. If this parameter is set (on) then the resulting grammar is not required to have an overt subject (e.g., Spanish). If it is not set (off) then the resulting grammar is required to have an overt subject (e.g., English). The starting state for each parameter should not be presupposed to be off. For each grammar there is a unique set of parameter values that identifies it. Learners in the principles and parameters framework are required to find the values for each parameter that will identify the target language. Once the parameters are set correctly then learning is complete. Chomsky developed the principles and parameters framework in part because it appeared to have an advantage over rule-based, transformational paradigms. It was envisioned that the amount of information that a learner would need to acquire intuitively seems small (linear with the number of parameters) in comparison with the amount of information that would be required to establish the correct rules and transformations from scratch. Some models deduce parameters values while others use probabilistic methods. All classes of languages in the principles and parameters

framework are guaranteed to be Gold-learnable because they all have a finite size. The rest of this section will describe some of the most important learners in the principles and parameters framework.

1.2.1 Triggers

1.2.1.1 Triggering Learning Algorithm

The first learner to be based on triggering was developed by Gibson and Wexler (Gibson and Wexler, 1994). They describe an algorithm which uses triggers to set parameter values within the principles and parameters framework. A trigger is a sentence which determines that a parameter must be set to a certain value in order for that sentence to be analyzed correctly or at all. The algorithm described is called the Triggering Learning Algorithm (TLA). The TLA proceeds as follows: Get an input sentence S. If it is recognized by the current hypothesis then move on to the next input sentence. Otherwise, uniformly select a parameter P and change its value. Call this new grammar G2. Now analyze S again using G2. If it is successful then G2 becomes the new hypothesis. Otherwise, keep the original value of the parameter which amounts to keeping G as the hypothesis. The TLA follows the error-driven, Greediness and Single Value Constraints (SVC) (Clark, 1992). The error-driven constraint says that the learner should only change its current hypothesis when it

cannot parse the current input sentence. The Greediness constraint means that the learner will only adopt a new hypothesis if the new hypothesis can analyze the current input sentence and the current hypothesis cannot. Finally, the Single Value Constraint says that at most one parameter may be changed per input sentence.

Triggers are divided into two types: local and global. A global trigger for a parameter value *v* of a single parameter P_i is a sentence from the target language such that the sentence can only be analyzed only if P_i is set to *v*. The values of the other parameters do not matter. A local trigger for a parameter value *v* of a single parameter P_i is a sentence from the target language such that the sentence can be analyzed only if P_i is set to *v* given a set of values for all other parameters. In contrast to global triggers, local triggers depend on the values of the other parameters.

Whether or not the TLA can converge on the target language in the limit depends on the existence of at least one local or global trigger for each incorrectly set parameter. It has been shown by Gibson and Wexler that at least one linguistically plausible parameter space exists for which there are no local triggers and therefore no way of reaching the target grammar.[5] They give two solutions to this problem. The first solution is to make the default initial grammar one such that a local maximum cannot be reached from it. The second solution is to initially delay the learner from setting parameters that can lead to a local maximum. They conclude that the existence of these solutions means that triggering theory may still be essentially correct (Gibson and Wexler, 1994, p 410). Gibson and Wexler prove that the TLA can converge but

[5] Learner is in local maxima.

not that it will necessarily converge. (Berwick and Niyogi, 1996) show that the TLA is not guaranteed to converge even if local triggers do exist for all languages.

It is important to note that the TLA is not guaranteed to learn languages that are subsets of other languages. Gibson and Wexler mention the existence of subset parameters, parameters whose different values result in subsets of one another. Since the TLA is error driven there can be no triggers for the subset value of a subset parameter if the learner has adopted the superset value. Due to this fact Gibson and Wexler restrict their discussion of triggers to language domains in which there are no subset-superset relations.

The TLA is important because it gave an algorithm for acquisition of grammars that are defined by a finite number of parameters. It has been the springboard for a great deal of research in computational modeling of parameter setting, (e.g., Berwick and Niyogi, 1996; Briscoe, 2000; Sakas, 2000a; among others).

1.2.1.2 The Deterministic Structural Triggers Learner

A major downfall of the TLA is that its version of triggering is nondeterministic. Given a current hypothesis and a trigger, all parameters are available for updating regardless of whether or not they will cause the current input sentence to be analyzable. Triggers in the TLA are ambiguous from the learner's perspective because the learner does not know which parameter value(s) need to be set in order to

accept the current input sentence.[6] In contrast, Fodor developed a deterministic learning device, the Structural Triggers Learner (*STL*) (Fodor, 1998b; Sakas and Fodor, 2001). The STL differs from the TLA in two important ways: (1) it detects and discards any input that is ambiguous as opposed to the TLA which uses it and (2) it only changes its grammar (i.e., adopts different parameter values) when it receives unambiguous input. The main 'rule' of the STL is "Do not learn from ambiguous input". The TLA on the other hand does not discriminate between parametrically ambiguous and unambiguous input. When the TLA receives an input that cannot be parsed by the current grammar it chooses a new parameter at random and changes the value. If the new parameter setting enables the grammar to parse the input the change is accepted. Otherwise, the parameter is retained at its current value. The problem here is that there may be more than one parameter change that would allow the input sentence to be parsed, the input is ambiguous. Setting a parameter on the basis of ambiguous input creates the possibility of setting the parameter incorrectly. If a parameter gets set incorrectly the TLA will either waste time resetting the parameter to the correct value or, even worse, get stuck in a local maxima and never attain the target language. The STL avoids this problem by using unambiguous triggers to drive the parameter setting process.

The STL is based on the idea that the parser should be used to identify triggers. Sakas and Fodor (2001) describe two main problems that need to be solved for triggering to work properly:

[6] TLA could set more than 1 parameter value if Single Value Constraint were removed.

1. The parsing paradox (Valian, 1990; Sakas and Fodor, 2001). The sentence processing mechanism can only parse those sentences licensed by the current grammar. Sentences that are not licensed by the current grammar contain information the learner needs to update the current grammar. The learner will never get the update information and as a result will never update the current grammar since the sentence processing mechanism cannot process the sentences it needs to extract information for the learner.
2. Parametric ambiguity. What parameters should a learner set for a sentence that is licensed by conflicting parameter values?[7]

The TLA is able to solve the parsing paradox by testing alternative grammars. However, it does not escape the problem of parametric ambiguity because it only tries one alternative per input sentence. When ambiguity is high the chance of adopting the wrong grammar increases. If the wrong grammar were chosen then the TLA would need a substantial number of sentences before encountering enough triggers that would allow the TLA to recover from this error. There is also the chance that it may never recover due to the presence of local maxima. Even if there were no local maxima in the domain, since the TLA is non-deterministic the randomly chosen parameter may very well create a candidate grammar that does **not** license the input sentence hence wasting a potentially informative input sentence. Due to greediness the learner will not adopt that grammar and the input sentence will have been wasted.

[7] Note that this is specifically a definition of *parametric* ambiguity. Another definition of ambiguity is when a sentence is parsable by multiple grammars. I.e., it is in more that one language. See Sakas & Fodor (2001) for discussion.

These conditions create a heavy workload for the TLA even for a small number of parameters.

The parametric principle (Sakas and Fodor (2001) following Chomsky (1981)) says that the value of each parameter should be established independently of the values of all others. The benefit of this is that for *n* parameters a learner only needs *n* pieces of information to find the target grammar. So the workload is linear with respect to the number of parameters. For this learning scheme to succeed, when a parameter is set there must be no doubt as to whether or not it has been set correctly. The TLA cannot guarantee that when it sets a parameter that it is in fact correct. The only time the TLA knows that any given parameter is set correctly is when it has attained the target grammar. The TLA always has a search space of 2^n grammars.[8] A learner that abides by the parametric principle would set each parameter once and that setting would be correct, effectively halving the search space with each parameter that is set, with the potential of dramatically increasing the speed of learning.

The original STL versions (Fodor, 1998b; Sakas and Fodor, 2001) solve both the parsing paradox and parametric ambiguity problems and also abide by the parametric principle. The STL uses the concept of structural triggers as a basis for learning. Structural triggers are small pieces of trees called *treelets* that are used both as a trigger and a parameter value. Treelets are presumed to be provided by the innate Univeral Grammar. Each treelet corresponds to a characteristic of a grammar. For

[8] But see Sakas (2000a) who demonstrates that under certain specific 'smoothness' conditions the TLA performs reasonably well despite the exponential search space.

example, there is a treelet that represents the complement-final value of the word order parameter for a verb phrase. If a grammar contained this treelet it would be complement-final for verb phrases. A grammar is just a combination of treelets (together with universal principles). The *Supergrammar* is all possible treelets (and universal principles).

One algorithm that makes use of structural triggers operates as follows,

Strong-STL Algorithm:

(1) Parse the current input with the current grammar.

(2) If the parse is successful then keep the current grammar and goto (1).

(3) Parse the current input with the supergrammar.

(4) If there is only one parse then adopt into the current grammar those treelets in the parse tree (drawn from the supergrammar) that are not in the current grammar and goto (1).

(5) If there is more than one parse adopt those treelets that show up in **all** of the supergrammar parses and goto (1).

(6) If there is more than one parse and there are no common tree fragments in the supergrammar parses then the input is fully ambiguous parametrically so retain the current grammar and goto (1).

The STL will never adopt a parameter value from an input if the input is ambiguous with respect to that parameter. In general, language domains with larger amounts of parametrically ambiguous sentences are harder to learn than language domains

containing smaller amounts of parametrically ambiguous sentences (see Sakas and Fodor (2001) for a detailed analysis of how parametric ambiguity affects learner efficiency). Information may be lost by dropping sentences but it is necessary to ensure error-free learning. The STL solves the parsing paradox by use of the "supergrammar" to parse any sentences not licensed by the current grammar. In this way it is guaranteed to always be able to parse a sentence even if it is not licensed by the current grammar. The STL follows the parametric principle because each treelet serves as a parameter value and that parameter value is adopted independently of the others. It will also only be set once as opposed to the TLA where it could be set many times.

For the STL, triggers are more than the left to right words that make up the input sentence; triggers are fragments of a tree **structure**. These fragments are considered to be innate and part of the Universal Grammar (UG). The Strong-STL is an "ideal" learner rather than a psychological model. It is useful as a standard against which to compare other models. However, Fodor and Sakas have been consistent in viewing the potentially massive parallel parsing that is required by the Strong-STL as psychologically implausible. The Strong-STL is an ideal learner in that it only sets parameter values if it knows those values are correct. Another STL variant called the Waiting-STL is also ideal in this respect. The Waiting-STL is the same as the Strong-STL except that it will adopt treelets only on the basis of sentences that have a single parse; i.e. fully unambiguous sentences. Whenever the Waiting-STL parser encounters a choice point during parsing, it notes that there is more than one possible

parse and chooses a path to complete the parse. This type of parsing is called a flagged serial parsing (Inoue and Fodor, 1995). Any sentence that results in a flagged parse will not be used to update the current hypothesis. In contrast, as noted above, the Strong-STL will compile every possible parse of the current input sentence and search those parses for treelets that appear in each one. Any treelets that appear in all parses are safe to adopt into the hypothesis grammar. An unambiguous trigger (either a sentence with treelet(s) appearing in all parses or a sentence with only one parse) paves the way for the creation of a deterministic learning algorithm.

In theory, both the Strong-STL and the Waiting-STL make it possible to conduct error-free learning in a reasonable amount of time for a parameter space large enough to describe natural languages. However, as I discuss below, large amounts of ambiguity in the language domain probably make these STL versions infeasible as models of human language acquisition. A variant of the STL called the Guessing-STL, which has a probabilistic component, works better in practice (Sakas and Nishimoto, 2002; Fodor and Sakas, 2004; Fodor and Teller, 2000).

1.2.2 Stochastic Methods

Stochastic methods use probabilistic algorithms to drive learning. Two important stochastic methods are Clark's Genetic Algorithm model (Clark, 1992) and Yang's Naïve Parameter Learner (Yang, 2002). The main advantage of stochastic methods is that they are able to perform hill-climbing searches with the benefit of being able to

escape from local maxima. In addition, there is no need to apply the subset principle to these models because they are not susceptible to the subset problem in the first place. It is important to have an understanding of these models because they show alternative methods for creating learners that can deal with the subset problem.

1.2.2.2 Genetic Algorithm Learner

Clark created the first statistical approach to parameter setting in the principles and parameters framework (Clark, 1992). Clark's learner uses a genetic algorithm (GA) to move through the search space. The learner starts by generating a population of hypothesis strings at random. Each hypothesis string is made of 0's and 1's that correspond to the values of each parameter. Next, each hypothesis string is compiled into a parsing device that represents that hypothesis string. This parsing device is an implementation of the parameter settings in the hypothesis string. An input sentence is now read in from the environment and parsed by each of the parsing devices. The parsing data is used as input to a fitness metric that determines how "good" the hypothesis is at parsing the input sentence. The fitness metric takes into account grammatical violations in the parse, subset relations, and general elegance of the parse. The subset and elegance factors are weighted such that they will not affect the computation as much as grammatical violations will. The genetic operators are now applied. A crossover operation is done on two hypotheses selected at random from the hypothesis strings. The random selection is weighted according to the fitness of the

hypotheses. More fit hypotheses are more likely to be selected for crossover than less fit hypotheses. A crossover is done by taking the first half of the parameters from one of the hypotheses and combining it with the parameters from the second half of the other hypothesis. The second half of the first hypothesis is also combined with the first half of the second hypothesis. For example, suppose the domain has four parameters and the two hypothesis strings are 1111 and 0000. The crossover operation for these strings would result in the hypothesis strings 1100 and 0011. The two new hypothesis strings are now added to the population. The learner now performs a mutation operation. A mutation operation is performed by selecting a hypothesis string at random from the population (according to fitness) and flipping one of its parameter values. For example, a mutation operation on the string 1111 could result in the string 1101. The final genetic operation is to eliminate hypothesis(es) from the population. Hypothesis(es) are chosen at random according to their fitness and removed from the population of hypothesis strings. The least fit hypothesis(es) are more likely to be eliminated than the most fit. The cross-over and mutation operations occur for each input sentence while the elimination of hypothesis(es) is only done occasionally. If the population consists of a single hypothesis string that matches the target then learning is done. Otherwise, start the process over by recompiling the population hypothesis strings into parsing devices.

The GA learner is not susceptible to the superset problem like error-driven learners are. The fitness metric will tend toward favoring hypotheses that generate smaller languages, just as the Subset Principle says to do. Supersets of the target may be kept

as part of the population for a while but they will eventually be deemed unfit and removed. In contrast to the GA learner, error-driven learners must never hypothesize a superset of the target because they have no way of updating the current hypothesis if it succeeds in parsing the current input sentence. For the GA learner, target languages that are supersets of other languages might seem like a problem because of the GA learner's tendency toward favoring smaller languages but they are not. If the target language happens to be a superset of some other language it will eventually be attained because the grammatical violations piece of the fitness metric will dominate the superset piece causing the GA learner to give up the smaller language. This domination will allow the superset hypothesis strings to prosper and eventually converge on the target.

1.2.2.2 Naïve Parameter Learner

Yang created a stochastic learner called the Naïve Parameter Learner (NPL) (Yang, 2002). The main idea of the NPL is that it rewards grammars that perform well and punishes grammars that perform poorly. Repeated applications of reward and punishment should ultimately guide the learner to the target grammar.

The NPL maintains a set of weights each of which corresponds to one parameter. The NPL starts by choosing a hypothesis grammar at random. The new grammar is created by randomly selecting values for each parameter according to their individual weights. All weights are initialized to 0.5. An input sentence is read in and tested

against the current hypothesis grammar. If the current hypothesis can parse the input then the learner rewards all the parameter values in the current hypothesis. Otherwise, it punishes all the parameter values in the current hypothesis. Rewarding a parameter means updating its weight such that it moves closer to its marked state. Punishing a parameter means updating its weight such that it moves closer to its unmarked state. Learning continues until all parameter weights are within a given threshold of either the marked or unmarked values. For example, if the threshold value is .001 then learning will continue until all parameter values are either less than .001 or greater than .999. The amount to punish or reward parameter weights can be changed so as to speed up or slow down movement through the search space. A large increment will allow for bigger jumps in the search space. This opens up the possibility of converging quickly on the target but it also makes the learner susceptible to oscillating parameter values that would hinder convergence. A small increment moves more slowly through the search space but it should always be moving steadily toward the target grammar.

The NPL does not employ the subset principle as part of its logic. It relies on the probabilistic nature of the algorithm to avoid superset hypotheses. If a situation arises where the superset value is set and it keeps getting rewarded then the learner will move towards the superset hypothesis. The NPL learner is susceptible to superset errors due to the fact that learning stops by threshold. If the set of parameter weights represent a superset hypothesis then all input sentences will reward those parameter values. As the parameter weights get closer to thresholds representing superset values

there is less of a chance that a non-superset grammar will be chosen at random. For the learner to escape from a potential superset error it would need to randomly select a grammar that is not the superset grammar and have that grammar fail to parse the current input sentence. As a result, the parameter weights will get punished and their values would move away from the superset hypothesis. The learner could still have its parameter weights move away from a superset hypothesis but the chances of that happening become smaller the closer the parameter weights are to a superset grammar. The time it would take to recover would depend on the increment of reward/punishment being used. A very small increment could essentially stop the learner from escaping the superset hypothesis. A large increment would allow the learner to escape much more easily. If the parameter weights become very close to a superset hypothesis then random grammar selection will settle on the superset hypothesis most of the time. The weights need to move from the superset hypothesis parameter values in order to increase the chance of generating hypotheses other than that particular superset hypothesis. A smaller increment requires more successive punishments of parameter weights that are close to the superset hypothesis as compared to a larger increment. A small increment value only changes the parameter weights a little each time so more punishments are needed to create a significant change. For the parameter weights to move from the superset hypothesis the parameter values need to be chosen that are different from the superset hypothesis. The likelihood of randomly choosing parameter values that do not make up the superset hypothesis is very low when the weights are close to those of the superset hypothesis.

1.2.2.3 Guessing STL

The Guessing-STL is a probabilistic learner that is a variant of the original STL (Sakas and Fodor, 2000; Sakas and Nishimoto, 2002; Fodor and Sakas, 2004). The Guessing-STLs work by choosing a parse when ambiguous input is encountered. This is in contrast to the Strong-STL and the Waiting-STL deterministic variants that discard the input if it is ambiguous.[9] The Guessing-STLs vary from each other by the strategy they use to choose a parse. The Any Parse strategy tells the learner to choose a parse at random. The Minimal Connections strategy chooses the parameter value that results in the smallest parse tree. The Least Null Terminals strategy selects the parse with the fewest empty categories. The Nearest Grammar[10] strategy chooses the grammar that differs least from the current hypothesis. Other guessing strategies could be implemented. The performance of the Guessing-STLs turned out to be better than the performance of the deterministic STL versions (see discussion below in Model Comparisons).

1.2.2.4 Probabilistic Components of the TLA

[9] Waiting-STL discards ambiguous input immediately while Strong-STL will try and find common treelets among parses and then discard if none are found. See section 1.2.1.2 for description of Waiting-STL and Strong-STL.
[10] Nearest Grammar is called Strong Oracle in Sakas and Nishimoto (2002).

Gibson and Wexler's Triggering Learning Algorithm also has a probabilistic component but it does not reap the rewards. In the TLA, the learner chooses a parameter at random when the current grammar hypothesis fails to parse the current input sentence. This randomness is not enough to help the TLA escape local maxima though. The randomness in the TLA is only encountered when the input sentence cannot be parsed by the current hypothesis grammar. If the TLA hypothesizes a superset of the target it will never recover because it is error driven and it will never change its hypothesis. The randomness in the TLA is subservient to the error driven nature of the learner. In each of Clark's and Yang's learners the probabilistic pieces are more prominent.

1.2.3 Feasibility

Much research has been devoted to the learnability of models (Gold, 1967; Angluin, 1980). Learnability is concerned with which language domains logically can and cannot be learned using a given learning model. The problem of feasibility has not been tackled until recently, however. Feasibility deals with how long it will take to learn languages in a domain as opposed to if the domain can be learned at all. It is important to know if something can be computed but it is also relevant to know how long it will take given that it can be computed. Methods of determining algorithm feasibility are useful in determining whether or not those algorithms are viable as a model of human acquisition. There would be no point in constructing models of human acquisition that are intractable. Methods of determining algorithm feasibility

are also useful as a means of comparison between algorithms as well as a way to uncover unforeseen problems with regards to local maxima (e.g., Niyogi and Berwick, 1996).

1.2.3.1 Learning From Triggers

Berwick and Niyogi (1996)[11] show that it is possible to model any memoryless learner as a Markov chain. In addition, they apply this technique to Gibson and Wexler's Triggering Learning Algorithm (TLA) (Gibson and Wexler, 1994). They describe more initial-final grammar pairs for which the TLA learner does not converge on the target and also describe flaws in the way Gibson and Wexler define a "problem state". Sakas (2000a; 2001) also applies Markov techniques to analyze the TLA as well as Fodor's (1998b) STL model. See discussion below.

Modeling the behavior of the TLA as a Markov chain makes it possible to estimate the average number of sentences required to learn a target language.[12] Berwick & Niyogi (1996) set up the Markov chain as follows:

Each grammar state represents a node. There is a link from one grammar (A) to another grammar (B) if (1) the hamming distance of the parameter settings is 1 and

[11] Berwick and Niyogi (1996) and Niyogi and Berwick (1996) contain much overlapping material. Since it is the overlapping material that is relevant to my dissertation, I will take the liberty to cite either of them.

[12] Sakas (2000a, p 30) gives a detailed description of how to perform the calculations for the average number of inputs required to learn a target language using a Markov chain.

(2) there is an input sentence that is a member of B that is not a member of A. Absorbing states are nodes with no outgoing links. These are either the target state or one of the local maxima (if any). All outgoing links have a transition probability associated with them. The sum of all outgoing links of a node must be 1.

Gibson and Wexler define a "problem state" as a grammar from which there is no path to the target grammar. A learner hypothesizing this state is destined to fail with probability equal to 1. Berwick and Niyogi show that this definition leaves out problematic grammars from which a learner *may* get to the target grammar, but from which a learner may *not* (with a probability less than 1). Berwick and Niyogi define a "problem state" as a grammar which is connected to a non-target absorbing state (or local maxima). They show that the probability of reaching these local maxima is significant and therefore that the Gibson and Wexler definition is flawed. So out of the 56 initial-final grammar pairs, Berwick and Niyogi calculate 12 that are not learnable with a probability of 1 as opposed to the 6 that Gibson and Wexler calculate.

Berwick and Niyogi also mention that the Gibson and Wexler maturational solution to the local maxima problem is not correct. They say that there is a significant probability that a local maximum could be reached even if the learner avoids early setting of a parameter that leads to a local maximum. G & W's maturational solution is based on the assumption that all strictly absorbing states are avoided by waiting for a finite period of time. Unfortunately, other absorbing states exist which can be

reached with a probability greater than 0. The existence of these other absorbing states means that G & W's maturational solution is not guaranteed to work in all cases.

Finally, they also show that the TLA's two heuristics (SVC and Greediness) slow learning down when combined. When one or the other heuristic is removed, the TLA surprisingly converges faster (cf., Sakas, 2003). This is significant because with either one of the two heuristics removed, *all* local maxima disappear.

1.2.3.2 Ambiguity and the Computational Feasibility of Syntax Acquisition

Following Niyogi and Berwick, Sakas (2000a) created a slightly different computational framework for analyzing the feasibility of parameter setting models of language acquisition. The performance of the TLA and the STL[13] were analyzed using this framework.

The computational framework is set up similarly to the one created by Niyogi and Berwick (1996). One difference in the frameworks, though, is the definition of a state when setting up the Markov structure. Niyogi and Berwick assign one grammar per state. The current formulation partitions all the grammars according to hamming distance from the target grammar and then assigns one partition per state. These partitions are called G-Rings. Sakas assumed that all grammars in a G-Ring have an

[13] Waiting-STL variant.

equal probability of parsing an input sentence (Weak Smoothness Requirement). Arcs represent the probability of moving from one G-Ring to another. Sakas defined a function that gives the probability of moving from one G-Ring to another. This function takes into account various factors such as ambiguity and parameter expression rate. Ambiguity is the number of grammars that can parse a given sentence. Parameter expression rate is the average number of parameters expressed by sentences of a given language. By varying these function parameters different learning environments can be simulated. A Markov structure is created from this state space graph. This Markov structure is used to calculate an estimate of the number of sentences required to learn a target language (Sakas, 2000a).

The analysis of the TLA shows that it performs best when the ambiguity is distributed according to a smooth domain. A smooth domain is one in which more similar grammars generate more similar languages. Sakas' Strong Smoothness requirement says that the domain should be weakly smooth (see above) and that the probability of a successful parse is greater for G-Rings closer to the target than for those G-Rings further away. In all other situations the feasibility of TLA performance is unreasonable, so much so that it is worse than a learner that chooses grammars completely at random.

Sakas (2000a) also performed an analysis of the STL. The STL seems to perform best in a realistic natural language domain when the parameter expression rate varies for each sentence. The STL performs well when the expression rate is fixed and low but

this does not simulate a realistic learning situation (cf. Sakas and Fodor, 1998). In order to simulate realistic conditions the expression rate must be high which in turn presents problems for the STL. These problems result from the fact that the deterministic STLs require parametrically unambiguous input to set parameters. Having a high parameter expression rate increases the chances that an input sentence will be ambiguous and consequently discarded. If the parameter expression rate were allowed to vary then it opens up the possibility that a sentence with a low expression rate will be encountered and that parameters will be set. The STL abides by the parametric principle (set individual parameters; don't evaluate whole grammars) which needs only a linear number of learning events as opposed to an exponential number that would be needed if the learner evaluated individual grammars. Once a few parameters are set, each subsequent input will be less parametrically ambiguous and successful learning events will occur more frequently.

Sakas' research creates a means of comparing different learning algorithms and grammar spaces according to their feasibility. This research is significant because it provides a method that can be used to analyze realistically sized natural language domains (which is not tractable using Niyogi and Berwick's model) and is not bound by the idiosyncrasies of any particular language. Sakas' model is more abstract and allows for the setting of simulation parameters that can be used to model a wide variety of learning environments. Further, it allows analysis of different sources of learning difficulty, which is not possible using Niyogi and Berwick's formulation.

The framework outlined in this research does not take into account domains containing languages in subset-superset relationships. Given any two languages in the domain a constraint is imposed that guarantees that there is at least one sentence in each language that is not in the other. The analysis would become much more complex if subsets were allowed. Superset avoidance or recovery mechanisms would need to be built into the learning algorithms and the cost of these procedures would not be reflected in the feasibility results of the framework as it is currently constituted. Learning algorithms exist which are inherently immune to the subset problem but there are others that are not. Comparisons of the feasibility of algorithms in those two groups are very important but would not be accurate because either (1) the learnability of algorithms susceptible to the superset hypothesis might be in question or (2) the cost of superset avoidance or recovery might not be reflected in the feasibility results. Suppose two algorithms are being compared. One algorithm, call it A, is probabilistic and can escape from a superset hypothesis (e.g., NPL). The other algorithm, say B, cannot escape from a superset hypothesis (e.g., error-driven with random grammar selection). Algorithm B could have better feasibility statistics as compared to algorithm A, which on the surface would make it seem like algorithm B is better than A. What is missing though is that the learnability of algorithm B should be in question. There is no way to tell from the feasibility results that algorithm B is susceptible to superset errors. Now suppose that algorithm B was altered such that it was able to avoid superset hypotheses. Such a mechanism might require work for the learner that is not accounted for in the feasibility calculations.

Algorithm B might have better feasibility results but in actuality be slower because of the superset avoidance mechanisms that were added but not accounted for.

1.2.3.3 Modeling the Effect of Cross-Language Ambiguity on Human Syntax Acquisition

Sakas (2000a; 2000b) presents a computational framework used to model the process by which human language learners acquire the syntactic component of their native language. The main focus of this model is to analyze the effect that parametric ambiguity has on the performance of a learner. Parametric ambiguity occurs when a sentence is licensed by more than one language. Parametrically ambiguous sentences are troublesome because they force the learner to choose between at least two different sets of parameter values, only one of which is correct. If the wrong one is chosen learning will be prolonged and in the worst case will not happen at all. The Sakas model judges a learner according to its feasibility or the amount of work (number of sentences) it takes to converge on the target language.

The number of input sentences consumed is derived by a Markov analysis. A Markov chain is set up such that states represent the number of parameters that have been set (as opposed to representing grammars, see discussion above) and state transitions represent the probability that the learner will set some number (w) of new, currently unset, parameters. The expression rate (e) is the average number of parameters expressed per sentence. The effective expression rate (e') is the average number of unambiguously expressed parameters per sentence. For an unambiguous domain,

Sakas shows that the probability that *w* new parameters will be adopted is hypergeometrically distributed. He folds the effective expression rate, the proportion of unambiguously set parameters, to the total number of parameters expressed. The use of the number of set/unset parameters together with the effective expression rate are the principal mechanisms used for formulating transitional probabilities between states. The function that calculates the expected number of input sentences uses the transition probabilities.

An example analysis was done on the Structural Triggers Learner (STL). The specific STL variant used was called the Waiting STL. The main strategy for the Waiting STL is that it should only learn from unambiguous input. Any ambiguous input sentences that are encountered are ignored (at least from the perspective of learning). For the waiting STL, ambiguity has its largest effect during the early stages of learning. STL performance improves dramatically after the early stages are done. More parameters are set so there is less ambiguity as learning progresses. Sakas showed that the STL is particularly susceptible to ambiguity because it will only learn from sentences that are unambiguous. However, results also showed that increasing the total number of parameters that need to be set had only a small effect on the amount of work that needed to be to be done. This is in strong contrast to the TLA model in which the amount of work needed to achieve the target grammar rose exponentially in the number of parameters that needed to be set. Subsequent research (summarized in Fodor and Sakas, 2004) attempts to keep the benefits of the STL scalability, while

speeding up learning by loosening the strict implementation of the Parametric Principle (see discussion in the next section).

Sakas also puts forth the idea that this model should be used in conjunction with a computational psycholinguistic study and a computer simulation. Comparison of the TLA and the STL most probably points to the fact that different models of acquisition are affected differently by different distributions and amounts of ambiguity. A computational psycholinguistic study should be used to determine the "shape of ambiguity" of natural languages, and see if the domain of languages used by the computer model of learning match those of the distribution of ambiguity in natural language. A computer simulation will empirically test the learner in question to establish its feasibility.

It is important to note that the domains used for the simulation of learners in all the studies cited above avoided subset-superset languages by removing all such relationships prior to the beginning of the simulation run. It is part of my ongoing research (with Sakas & Fodor) to investigate to what extent parametric ambiguity is relevant to the subset problem. It is an open question how large a cost a learner may incur in a domain with subset/superset relationships and how the amount of parametrically ambiguous input is related to that cost. Knowledge of the distribution of parametrically ambiguous input due to a subset/superset relationship in conjunction with the cost of subset/superset specific logic could be valuable information when determining the feasibility of a learner that incorporates subset/superset logic.

1.2.3.4 Model Comparisons

Research was done that compares different search heuristics used to guide learning in the principles and parameters framework (Sakas and Nishimoto, 2002; Sakas, 2003; Fodor and Sakas, 2004). The heuristics are judged according to their feasibility (the time/effort, measured in terms of number of input sentences required, it takes to attain the target grammar). The heuristics used in these studies were partitioned into four different basic algorithms and their variants: Error-Driven Blind Guess (EDBG), Triggering Learning Algorithm (TLA) (Gibson and Wexler, 1994), the Variational Learner (VL)[14] (Yang, 2002) and the Structural Triggers Learner (STL) (Fodor, 1998b; Sakas and Fodor, 2001). The heuristics can further be divided into two main categories: those that guide the learner according to a can parse/cannot parse outcome and those that guide the learner according to parse tree information. The EDBG, TLA and VL exclusively use can parse/cannot parse information while the STL variants make use of both.

In Sakas and Nishimoto (2002), results on a small 4 parameter domain, showed unsurprisingly, that the STL strategies (Strong-STL, and the Nearest Grammar Guessing-STL variants) which availed themselves of a maximum amount of structural information which is obtained through a full parallel parse of every input

[14] The variational learner is a variant of Yang's Naïve Parameter Learner (NPL).

sentence. The Waiting-STL, which discards all sentences that contain ambiguous information, performed the worst of all the STL variants.

The Variational Learner (Yang, 2002), which uses statistics over a can parse/cannot parse outcome performed even worse than the STL models. Sakas & Nishimoto note, however, that the Variational Learner works well at the outset of learning in a highly ambiguous domain. In conclusion, they conjecture that the best approach might be a combination of the two main heuristics: Use a statistical heuristic such as the VL in the early stages of learning and then switch to a structural heuristic such as the STL when statistical learning starts to deteriorate. The STL operates most efficiently after some parameters have been set. By using the VL in the beginning and the STL from then on, the performance of the learner can be maximized for the duration of the simulation. [15]

Again, for all of the studies discussed in this section, none of the search heuristics directly address the subset problem. Instead, simulations were implemented by removing all supersets of the target language prior to the each simulation run. Leaving the superset languages in the domain, and studying the performance of strategies that actively apply the Subset Principle during the course of learning, is the focus of this thesis.

[15] Fodor (1998a) has a similar model to the Variational Learner, the Parse Naturally STL, which counts how often a parameter value was used in a successful parse and picks a grammar based on those counts. Though untested by Sakas and Nishimoto, Sakas is optimistic that this model might well perform best of all the STL variants (Sakas, pc).

1.3 The Subset Principle

The Subset Principle is a rule that learners can follow that would allow them to avoid hypothesizing languages that are supersets of the target language. Explicit formulation of the Subset Principle is generally credited to Berwick (1985) and Manzini and Wexler (1987) and Wexler and Manzini (1987). However, to the best of my knowledge, Gold (1967) was the first to mention problems related to learning classes of languages, from positive data only (see section 1.1.1 Identification In The Limit), and Angluin (1980), among many others, continues investigation in Gold's paradigm in which the problem of "overgeneralization" remains a central concern. While the Subset Principle is generally accepted and deemed relevant by a large segment of the research community, there are those who believe that subset languages do not exist in the domain of natural (human) languages, and that the Subset Principle is not a necessary aspect of human language learning. This section will start with a discussion of the research by Berwick (1985) and Manzini and Wexler (1987) in the formulation of the Subset Principle. Next, some research that argues against any need for the Subset Principle will be reviewed. Finally, research by Fodor and Sakas (2005) will be reviewed. Their article goes into problems that arise up if the Subset Principle is faithfully obeyed by a learning algorithm that has psychocomputationally plausible restrictions on what (and how much) can be retained in the memory store.

1.3.1 Subset Principle Background

1.3.1.1 Berwick

Berwick (1985) states that the Subset Principle is necessary (though not sufficient) for identifiability from positive evidence. He states the Subset Principle informally as follows: “Briefly, the Subset Principle states that learning hypotheses are ordered in such a way that positive examples can disconfirm them.” If the hypotheses were ordered in some other way then positive only evidence would not be enough for the learner to attain the target grammar. The learner could hypothesize a superset before the target and the desired final acquisition of the actual target language would never take place. All sentences would be grammatical with respect to the superset language and the learner would never change its hypothesis and consequently never converge on the target language, hence the need for the Subset Principle. Berwick’s formulation of the Subset Principle was based on research done by Gold (1967) and Angluin (1980).

Berwick also mentions that the problem of determining whether one language is a subset of another language is in the general case undecidable. (Though Joshi (1994) has shown that it is decidable over the tree sets of context-free grammars.) This is relevant because the learner needs to know when to apply the Subset Principle. If the

learner could not determine subset/superset relationships then acquisition cannot be guaranteed. Berwick (1985, p 237) concludes that either the ordering of languages was determined as humans developed as a species due to natural selection (i.e., it's innate, also see discussion in Fodor & Sakas (2005) section 2) or that the calculations are tractable due to constraints on natural languages such as a limited depth of recursion.[16]

1.3.1.2 Manzini and Wexler

Manzini and Wexler argue for the existence of subset languages in the natural language domain and the implications of their existence for a language learner (Manzini and Wexler, 1987; Wexler and Manzini, 1987). They construct examples within binding theory to show that the existence of subset languages is linguistically reasonable. The Subset Principle is defined and used as the main component to drive learning in a domain that contains subsets.

Manzini and Wexler use Binding Theory to give examples of subset languages. They assume that Principles A and B as defined by Chomsky (Chomsky, 1980; Chomsky, 1981; Chomsky, 1982) hold. It is shown that different values of the governing category parameter for anaphors generate languages that are subsets of one another. For example, assume that value 1 of the governing category parameter says that the

[16] Berwick cites research by Wexler and Culicover (1980) showing that a transformational grammar is learnable from input sentences with a maximum of two embedded clauses.

governing category for an anaphor *a* must contain *a*, a governor for *a*, and a subject. Also, assume that value 2 of the governing category parameter says that the governing category for an anaphor *a* must contain *a*, a governor for *a*, and an inflection. Given other principles of Government and Binding Theory, it follows that all categories with an inflection are categories with a subject, but not all categories with a subject are categories with an inflection. So with respect to the distribution of anaphors, the languages that have value 1 of the governing category parameter are a subset of the languages that have value 2 of the governing category parameter. The examples given by Manzini and Wexler contain more values for the governing category parameter but the idea is the same. The inclusion hierarchy described for anaphors is also shown for pronominals except that the subset/superset relationships go in the opposite direction.

The Subset Principle is necessary for learning a parameter value when the languages generated by the parameter are in an inclusion hierarchy. If the languages are not in an inclusion hierarchy then some other method can be used to select the parameter value. M&W define the Subset Condition which states that for any two values p_i and p_j of a parameter p, either the language generated by the value p_i is a subset of the language p_j or vice versa. The values of the other parameters remain fixed. If the Subset Condition holds for a parameter then the learner must use the Subset Principle to select the value of that parameter. If the Subset Condition does not hold then some other method should be used to select the value of that parameter.

Manzini and Wexler also define another principle, the Independence Principle. In M&W's framework, both the Independence Principle **and** the Subset Condition must hold if the Subset Principle is to be invoked successfully. The Independence Principle states that if the language generated by value p_i of parameter p is a subset of the language generated by value p_j of parameter p that relationship will always hold regardless of the values of any of the other parameters (as long as the values of the other parameters remain fixed when comparing the subset and superset values of p). For example, if 011 is a subset of 111 with p_1 being a subset parameter then we can also say that 001 is a subset of 101. Notice that the values of the other parameters remain constant when comparing values of the subset parameter. M & W do not make any statements about what happens if other parameters are allowed to vary at the same time as the subset parameter (but cf. Fodor and Sakas (2005) definition of independence). The Independence Principle is important because it allows the learner to use the Subset Principle to set the values of parameters individually.

This research was important because it gave examples of languages that were subsets of one another and how a learner could learn in such a domain. Much research has been done in response to the ideas put forth. Some research questions the existence of such languages (Kapur, Lust, Harbert et al., 1993) while other research supports it (Wexler, 1993). The generally prevailing opinion seems to be that some but not all natural language parameters create subset-superset relations. It is an open area of research to create learners that abide by the subset principle. Major learning models have been created since this work in 1987, but the authors only mention the existence

of subset languages and then set aside the problem (Gibson and Wexler, 1994; Fodor, 1998b). The implication of creating a learner that does abide by the Subset Principle has been addressed, though, by Fodor and Sakas (2005).

1.3.2 Arguments Against the Subset Principle

1.3.2.1 Becker

Becker (2006) puts forth an argument against using the Subset Principle for learning raising and control verbs. The reasons for this are that (1) ambiguous verbs could cause the learner to incorrectly set the verb class parameter and that (2) children seem to learn in a way that assumes the wrong default parameter value needed for using the Subset Principle. It is suggested that a probabilistic approach using cues for raising and control verbs is a better learning strategy than the Subset Principle.

The significance of this research is that if it is correct it gives support to the probabilistic learning paradigm (e.g., Yang, 2002) over the triggered learning paradigm (e.g., Gibson and Wexler, 1994; Fodor and Sakas, 2004). There may be ways around it such as along the lines suggested by Gibson & Wexler (1994) for the verb second parameter. That is, setting the marked value of the verb class parameter may be delayed for a period of time, which would allow the triggered paradigm to work.

1.3.2.2 MacLaughlin

MacLaughlin (1995) critically examines the need for the Subset Principle in first language acquisition as a means of determining its applicability to second language acquisition. MacLaughlin concludes that the subset learning problem, which the Subset Principle is intended to solve, does not arise in the first place. She argues that the Subset Principle does not apply in situations in which it is standardly assumed to apply. Basically, her arguments involve reorienting the linguistic descriptions of syntactic parameters. For example, the Case Adjacency parameter is related to the adjacency of a verb to its object. A side effect of the case adjacency parameter is to regulate where adverbs can appear within a verb phrase. A grammar with strict adjacency such as English does not allow an adverb to appear between a verb and its direct object. French on the other hand does not have strict adjacency and it does allow an adverb to appear between a verb and its direct object. A grammar without strict adjacency can parse sentences in which verb and object are strictly or not strictly adjacent so the Case Adjacency parameter can be thought of as a subset parameter. MacLauglin makes the argument that the effects just described would be better explained in terms of verb raising. In French, when the verb raises the adverb will appear between the verb and its direct object in the surface form. If the verb does not raise, such as in English, the adverb will appear to the left of it in the surface form. The verb raising puts the English and French grammars in an intersecting relationship as opposed to a subset/superset relationship. She also gives similar

arguments for the Pro-drop parameter and the Bounding node parameter.

MacLaughlin also brings up several points that she believes to be problems with the Parameterized Binding Theory of Manzini and Wexler (1987). These problems are similar in spirit to her arguments concerning the Case Adjacency parameter, the Pro-drop parameter, and the Bounding Node parameter.

1.3.3 Subset Principle Compliance

Fodor and Sakas (2005) discuss how the Subset Principle (SP) could be implemented so that it is feasible both linguistically and psychologically. A major point uncovered by their research is the fact that incremental learning and SP are incompatible. They also discuss incompatibility of the Single Value Constraint and SP. In addition, some new learning models that are faithful to SP are presented and problems with those models are discussed.

1.3.3.1 The Shifting Problem

Previous research (Clark, 1992) states that SP is not a solution to avoiding superset errors because of a 'shifting' problem between parameters. The following language domain, as depicted by Fodor & Sakas (2005, p 530) shows an example of Clark's original example of shifting problem.

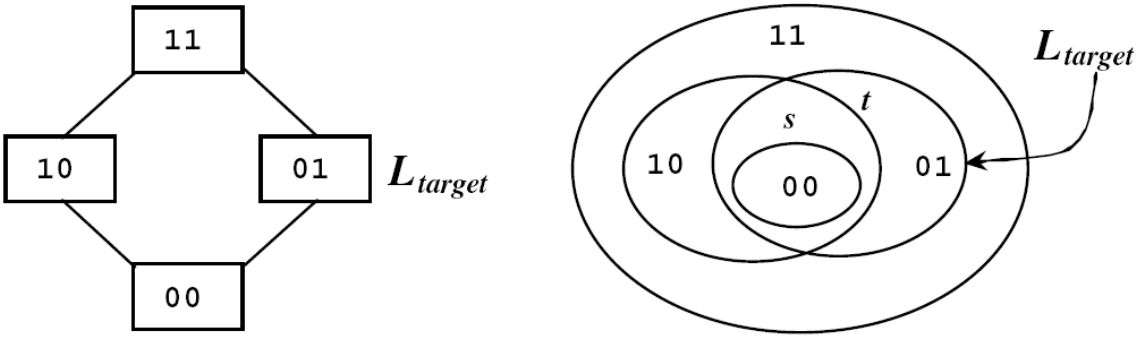

Figure 1: Shifting problem.

The problem occurs when the learner hypothesizes 10 and the target is 01 (or vice versa). Clark assumes that the learner abides by the Single Value Constraint which says that the learner can only change one parameter at a time keeping the other parameters fixed. The SVC is psychologically attractive because it models a gradual time course. Given SVC, there is no way for the learner to move from 10 to 01. From 10, the learner can only move to 11, which would cause a superset error assuming the target is 01. Fodor and Sakas (2005) argue that the problem is not SP per se, but rather Clark's strict application of the SVC. If the learner were allowed to update two parameters at the same time when required to do so by a stronger constraint like SP it could move from 10 to 01.

Another issue with moving from 10 to 01 is that the learner must give up a marked value (i.e., the first parameter goes from 1 to 0). Fodor and Sakas refer to this as retrenchment – for the learner to be safe from a fatal superset hypothesis, the learner must always start from 'scratch'. In the case of parameters, this means resetting

parameters to their least marked values.[17] Using retrenchment and giving up the SVC would allow the learner to move from 10 to 01. Learning would proceed as follows: The learner starts at 00 and then hypothesizes 10 due to some input. 10 is then disconfirmed due to the next input sentence *s*. Assume also that *s* is not in the language generated by 00. At this point the learner retrenches to the least marked grammar that licenses *s*, which is 01, and learning is complete. In previous research SVC is given precedence over SP and that should not necessarily be the case. Fodor and Sakas say that if SP is used it should take precedence over all other constraints. Obeying SVC is fine as long as it does not affect application of SP.

1.3.2.2 The Simple Defaults Model

Following M&W, Fodor and Sakas describe a learning model called the Simple Defaults Model (SDM). The SDM was designed to add a level of prioritization to grammar hypotheses so that SP is always obeyed. In the SDM all parameters have a default value that is a subset of its marked value. Each parameter does not necessarily designate a subset/superset relationship, but if it does, the marked value must be a superset of the default or unmarked value. The SDM also requires that the only subset/superset relationships that exist in the language domain are exactly those that come about from changing an unmarked value of a subset/superset parameter to a

[17] Of course this assumes less than perfect information about the correct parameter settings for the target language. If the learner were able to determine absolutely that a marked parameter value was necessary to parse an input sentence (e.g., Strong-STL), then the learner would not need to retrench to the unmarked value of that parameter. So far this strong form of the parametric principle (Sakas and Fodor, 2001) has proven elusive under reasonable psychologically plausible computational constraints. However, see Sakas & Fodor (in prep.) for re-examination of the issues.

marked value. In addition, the default value and the marked value of a parameter must never reverse a subset/superset relationship over all combinations of the other parameter values, even if they are varying at the same time. This is Fodor and Sakas's variant of M&W's Independence Principle.

1.3.3.3 The Ordered Defaults Model

A variant of the SDM is the Ordered Defaults Model (ODM). The ODM is the same as the SDM except that the parameters are ordered in some manner. This ordering is necessary (but not sufficient) for a learner to succeed when a language domain does not respect the Independence Principle. In this case, the SDM would be susceptible to superset errors while the ODM would not be susceptible due to the ordering of parameters. The following language domain, as depicted by Fodor & Sakas (2005, p 526) is an example of a language domain that is learnable by the ODM but not the SDM.

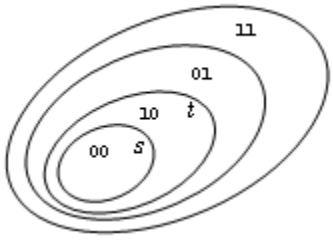

Figure 2: Example language domain learnable with ODM but not SDM.

The learner begins at grammar 00 and the target is grammar 10. The SDM would be susceptible to superset errors in this language domain because it does not have access to the knowledge that 01 is a superset of 10. From 00, and on receiving input t, the SDM could set the marked value of parameter p_2 and cause the learner to hypothesize 01. This is a superset error. In contrast, the ODM would not be allowed to set the marked value of p_2 before p_1. The ordering of parameter values by the ODM makes this language domain learnable. However, both the SDM and the ODM will fall prey to Clark's shifting problem if the SVC is strictly enforced (because retrenchment on p_1 from 10 to 01 is necessary when 01 is the target).

1.3.3.4 Effects of Incremental Learning

No memory for past inputs makes a learner susceptible to chronic undergeneralization errors. Undergeneralization means selecting a language that is smaller than the target. An undergeneralization error is the opposite of an overgeneralization or superset error. Even if the learner does receive an input that triggers a superset language close to the target, it still runs the risk of falling right back into a subset language on future inputs. The learner will undergeneralize with a high frequency because (given an incremental learning framework) it can only use the current input sentence to select the next language, and it must pick the smallest language compatible with that input (Fodor & Sakas, 2005). If the learner were allowed memory for past inputs this situation could be mediated because the smallest language compatible with several (or many) inputs is likely to be larger than the smallest language compatible with the

single current input sentence.

Fodor and Sakas also suggest modularizing the grammar as another solution to the undergeneralization problem. Using this solution the parameters would be grouped into different modules such as case, theta, binding, bounding, etc. The learner would only be allowed to reset parameters that are located in the same module. The effect of this would be to limit the amount of retrenchment that takes place which would in turn increase the speed of acquisition. Although attractive from the point of view of SP, it is probably not a linguistically viable option as parameters appear to interact across modules (see discussion in Clark, 1992). It seems that undergeneralization errors due to retrenchment can be just as harmful as superset errors.

1.3.3.5 Subset-free Triggers

Fodor & Sakas (2005) argue (though do not mathematically prove) that unless some other solution for excessive retrenchment can be found, the language domain for an incremental learner must be such that every language L contains at least one sentence such that L is a smallest language containing that sentence. This sentence is referred to as a subset-free trigger (sft). All languages in a language domain must contain a subset-free trigger in order for that language domain to be learnable.

1.4 The Simulation Platform

A simulation platform for first language acquisition was developed. The platform contains approximately 6000 lines of C++ code and can be run in either a Windows or Linux environment. Virtually all of the simulations are run on a Linux machine. The platform was designed to be easily extensible. New learners are created by deriving from an abstract base learner class and adding a few lines of code to the main Simulation class. The program output is in both tab-delimited format that can be easily read into an application for analysis (e.g., Mathematica, Microsoft Excel, etc.) or XML format which, in combination with XSL and CSS stylesheets, can be viewed using a web browser. Two other utility programs were also written in ASP.NET/C#. These utilities are used to query the language domain lattice of languages in order to assist in analysis of data and debugging.

1.4.1 CoLAG Domain

The simulation platform was designed to use the CUNY Computational Language Acquisition Group (CoLAG) domain (Fodor, Melnikova and Troseth, 2002; Sakas, 2003). The CoLAG domain was created in order to facilitate simulation research as opposed to standard learnability models which use proofs to give results. The CoLAG domain is based in the principles and parameters framework (Chomsky, 1981). It is a

database of word order patterns for 3072 abstract languages. The word order patterns were designed to reflect a wide range of natural language syntactic phenomena. There are 13 parameters or points of variation between languages in the domain. Learners that run on our simulation platform are presented with grammatical sentences from the target language (which can be any of the 3072 languages). All sentences of the target language are equally likely to occur. The goal of the learner is to hypothesize the target language given the input sentences.

1.4.2 Simulation Program

The simulation program was designed to facilitate easy creation of new learners. Inheritance and polymorphism were used extensively throughout the program as a means to isolate learner specific logic and factor out common logic. New learner creation essentially boils down to writing one new class.

All learners must be derived from the abstract base class Learner. The Learner class models the basic structure of what all learners running in our simulation framework must provide. All learners must be able to set their first hypothesis according to their given algorithm. A learner must be able to reset itself for the next trial. Finally, a learner must be able to pick its next grammar hypothesis according to its given algorithm. The PickNextHypothesis() procedure is where the bulk of the logic that makes each learner unique resides. Figure 3 contains the definition of the base learner class. All class and procedure definitions are written in pseudocode (see Appendix B

– Pseudocode Guide for pseudocode descriptions). Some programming details from the actual implementation are left out in order to make the code more readable.

```
ABSTRACT CLASS Learner
      CLASSPROCEDURES
            VIRTUAL Reset();
            VIRTUAL SetFirstHypothesis();
            VIRTUAL PickNextHypothesis();
      ENDCLASSPROCEDURES

      CLASSVARIABLES
            DECLARE HypoGrammID;
            /* HypoGrammID: Current hypothesis grammar ID.
      ENDCLASS
ENDCLASS
```

Figure 3: Learner base class.

The simulated learner is trained on each of the 3072 languages in the CoLAG domain. Each of those languages is run for a given number of trials, typically 100. All learners use the Simulation::Run() procedure to drive processing. The Run procedure provides the main processing loop for the simulation. The Run procedure is defined in Figure 4.

```
PROCEDURE Simulation::Run ()

DECLARE Learner;

/*    Learner:    Stores the instance of the simulated learner. */

Learner ← Simulation.CreateLearner();

FOR target IN targetLangs DO
      Environment.SetupTargetLanguage(target, LanguageDomain);
      Oracle.SetUp(target);

      FOR i FROM 1 TO NumTrials DO
            Learner.Reset()
            SearchSpace.ResetForTrial();

            Learner.SetFirstHypothesis();

            WHILE NOT Oracle.AttainedTarget(Learner.HypoGrammID) DO
                  Learner.PickNextHypothesis();
            ENDWHILE
      ENDFOR;
ENDFOR;

ENDPROCEDURE;
```

Figure 4: Main processing loop.

The global variables that the program uses are listed in Figure 5. These variables are accessible from anywhere in the program.

```
DECLARE     Simulation, SearchSpace, Environment, Oracle, LanguageDomain,
            Lattice;

/*          Simulation:         Drives the simulation program. The Run
                                procedure of this class actually executes the simulated
                                learner.
            SearchSpace:        All grammar hypotheses currently available to the
                                learner.
            Environment:        Encapsulates the learning environment. The target
                                language sentences are stored here.
            Oracle:             Responsible for deciding when processing should
                                stop. Stores the target grammar. Stores grammars that
                                are weakly equivalent to the target grammar. Stores
                                supersets of the target grammar.
            LanguageDomain:     All languages in the CoLAG domain.
            Lattice:            The lattice of all subset-superset relationships in the
                                language domain.
*/
```

Figure 5: Global variables.

Other procedure names appear in the Run procedure code and in the derived learner overrides of PickNextHypothesis. Descriptions of all of these procedures are given in Appendix A – Miscellaneous Procedure Descriptions.

The new derived learner class must provide its own implementations of the Reset(), SetFirstHypothesis(), and PickNextHypothesis() procedures. For example, a learner that simply chooses a hypothesis grammar at random would be defined as in Figure 6 through Figure 9.

```
CLASS ExampleLearner INHERITS Learner
     CLASSPROCEDURES
          Reset();
          SetFirstHypothesis();
          PickNextHypothesis();
     ENDCLASSPROCEDURES

     CLASSVARIABLES
     ENDCLASSVARIABLES
ENDCLASS
```

Figure 6: An example learner class.

```
PROCEDURE ExampleLearner::SetFirstHypothesis()

/* Choose a grammar hypothesis at random. */
HypoGrammID ← LanguageDomain.PickRandomGrammar();

ENDPROCEDURE;
```

Figure 7: Setting the first hypothesis for the example learner algorithm.

```
PROCEDURE ExampleLearner::Reset()

/* No reset code is needed for this particular learner.
*/

ENDPROCEDURE;
```

Figure 8: Resetting the example learner for a new trial.

```
PROCEDURE ExampleLearner::PickNextHypothesis()

DECLARE Sentence, CandGrammID;
/*    Sentence:         The current input sentence. */

/* Get the next input sentence from the linguistic environment. */
Sentence ← Environment.GetAnInput();

/* If current hypothesis can parse then just retain it. */
IF Licensed(HypoGrammID, Sentence)
      THEN RETURN;
ENDIF;

/* Choose a grammar hypothesis at random. */
HypoGrammID ← LanguageDomain.PickRandomGrammar();

ENDPROCEDURE;
```

Figure 9: Example learner algorithm.

The example learner override of PickNextHypothesis gets an input sentence from the linguistic environment and checks to see if the current grammar hypothesis can parse it. If it can parse the input sentence then the current hypothesis is retained otherwise a new grammar hypothesis is chosen at random from the language domain. It should be noted that this example learner is susceptible to superset errors because it does not obey SP when choosing a new grammar hypothesis.

2 Partial Ordering Learners

2.1 Justification For Adding Memory For Disconfirmed Grammars

The learners being created in the current research make a hypothesis about the target language after each individual input sentence. No negative or disconfirming evidence about the current hypothesis grammar is given to the learner. If a learner in this environment were to hypothesize a superset of the target language then it would be impossible for convergence on the target grammar to take place due to the absence of negative evidence. We are investigating learners that use the Subset Principle (SP) to avoid any superset hypotheses - overgeneralization. Faithful application of SP solves the overgeneralization problem but can create a chronic undergeneralization problem. The learner can become too conservative and possibly never hypothesize larger target languages. One goal of the current research is to incorporate memory for past grammars into the learner in order to avoid this undergeneralization problem. Adding memory to the learner will decrease the search space by eliminating languages in the domain as learning progresses and remedy the undergeneralization problem. But it remains an open question how much memory would be needed to produce a significant decrease (if any) in the amount of work necessary for the learner to attain the target language. For example, for a language domain of 30 parameters, eliminating one grammar at a time may not have much of an effect on the efficiency of the learner because the search space is so large.

2.2 General SP Lattice Learner

Gold (1967) posited a total ordering of grammar hypotheses with respect to subset/superset relationships to guarantee that the learner will not hypothesize a superset of the target. However, it is computationally unrealistic as a model for child language learning. If learners running on our simulation platform were to use a total ordering of grammar hypotheses then it would be necessary for them to hypothesize virtually all 3072 languages in the domain for those target languages that are located at the end of the ordering. The total ordering of grammar hypotheses guarantees that the learner will not hypothesize a superset of the target but it is too inefficient to be considered from a psychocomputational point of view (though see Fodor & Sakas (2005) for discussion). One possible solution to this problem would be to relax the restriction of a total ordering of grammar hypotheses to a partial ordering of grammar hypotheses.

The *General SP Lattice Learner* (Lattice learner) uses a lattice or partially ordered set of languages (POSET) to navigate the search space (Fodor, Sakas and Hoskey, 2007). The partial ordering of the language domain lattice (*LD lattice*) is based on the subset-superset relations of the languages in the domain. Throughout this paper, we envision superset languages being "above" their subsets in the lattice. The term "smallest" is also used to describe certain languages in the lattice. A smallest language is a language that does not contain any subsets. There are many smallest

languages in the CoLAG domain. Let SL be the set of smallest languages in the CoLAG domain. When the Lattice learner has to choose a new hypothesis it must make that selection from SL. All the languages in SL are smallest languages so the learner by definition is abiding by SP when it chooses from SL. Whenever the current hypothesis is disconfirmed that language is removed from the LD lattice and SL. The parents of that node may then be added to SL. The parents of the disconfirmed language cannot be blindly added though. All children of the parents must be checked to see if they have any other children to make sure they are in fact smallest languages. If the parents do have other children then they cannot be added to SL. This check guarantees that all languages in SL are smallest languages. The removal of languages from SL and the LD lattice is effectively giving the learner memory for past grammars. Given that all the learners are error-driven, previously hypothesized languages are effectively disconfirmed and should not be hypothesized again.

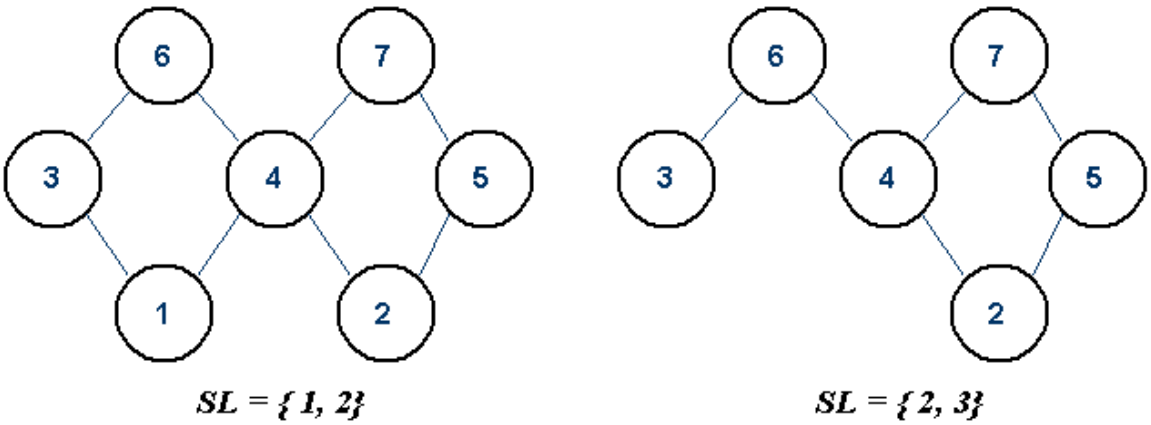

Figure 10: Remove language 1 from SL and the lattice.

```
CLASS SPLatticeLearner INHERITS Learner
      CLASSPROCEDURES
            Reset();
            SetFirstHypothesis();
            PickNextHypothesis();
      ENDCLASSPROCEDURES

      CLASSVARIABLES
            DECLARE Lattice;
            /* Lattice: Contains all the subset-superset
                        relationships of the grammars. */
      ENDCLASSVARIABLES
ENDCLASS
```

Figure 11: SPLatticeLearner class.

```
PROCEDURE SPLatticeLearner::PickNextHypothesis()

DECLARE Sentence, CandGrammID;

/*    Sentence:         The current input sentence.
      CandGrammID:      A candidate grammar hypothesis. The learner
                        is entertaining the idea of making this
                        grammar the next hypothesis. */

Sentence ← Environment.GetAnInput();

IF Licensed(HypoGrammID, Sentence)
      THEN RETURN;
ENDIF;

Lattice.Remove(HypoGrammID);

CandGrammID ← Lattice.PickRandomGrammarFromSmallestLanguages();

IF Licensed(CandGrammID, Sentence)
      HypoGrammID ← CandGrammID;
      RETURN;
ENDIF;

Lattice.Remove(CandGrammID);

HypoGrammID ← Lattice.PickRandomGrammarFromSmallestLanguages();

ENDPROCEDURE;
```

Figure 12: Lattice learner algorithm.

2.3 SP Lattice Learner Variants

All variants of the lattice learner abide by SP and use the lattice to help guide hypothesis selection. The parser is also used by each learner to varying degrees. See Fodor (1998a) and Sakas & Fodor (2001) for a detailed description of using the parser during hypothesis selection.

The SP Lattice Decode learner variants use the parser initially for each hypothesis selection but will ultimately be guided by SP constraints before the next hypothesis is selected. The Integrated learner uses the parser in conjunction with the lattice to guide hypothesis selection. The Flashlight variant of the SP Lattice learner uses previous hypothesis selection to help guide the current hypothesis selection. The Retrench learner uses the parser to find a starting point in the lattice for hypothesis selection. A detailed description of each of these learners will be given in the sections that follow.

2.3.1 Parallel Parsing Vs. Serial Parsing

From the perspective of computational load on the parser, we can separate parsing into two categories, parallel parsing and serial parsing. Parallel parsing means that the parser generates every possible parse of a given sentence. The parser is given the input sentence and it will continue processing until all possible parses are generated. It is unlikely that human language acquisition entails using parallel parsing because it is thought to be too computationally demanding a task. Serial parsing means generating one parse by selecting at choice points (if there are any) according to a given algorithm (e.g., select a path at random). Serial parsing does not generate as much information as parallel parsing does but it is much more feasible computationally.

2.3.2 Decoding

Decoding can be thought of as an intelligent version of serial parsing. The main difference between serial parsing and decoding is that decoding actively generates a parse while serial parsing does not. Decoding is the process of both parsing the current input sentence with the current grammar and if necessary patching additional unused treelets into the current grammar in order to complete the parse. Any new treelets are added to the parse tree at a point where parsing failed. The pool of treelets that the parser draws from is supplied by the supergrammar (ultimately from UG). Decoding makes it possible for the learner to generate a valid parse of the current input sentence when a normal serial parse would not. The grammar generated by decoding may not be the target but it will at the very least parse the current input sentence. Some learners may not use the generated grammar due to SP constraints but if they do then they will have added treelets to the current grammar that are a necessary part of the target grammar.

2.3.3 Decode Learner

The *SP Lattice Decode Learner* (Decode learner) uses decoding to guide hypothesis selection. The current input sentence is parsed with the current grammar. If the current grammar can parse the current input sentence then it is retained. Otherwise, the learner chooses one parse of the current input sentence at random and checks for

membership of the associated grammar in SL. If it is in SL then that grammar becomes the current hypothesis. Otherwise the learner retains the previous hypothesis.

```
CLASS SPLatticeDecodeLearner INHERITS SPLatticeLearner
      CLASSPROCEDURES
            Reset();
            PickNextHypothesis();
      ENDCLASSPROCEDURES

      CLASSVARIABLES
      ENDCLASSVARIABLES
ENDCLASS
```

Figure 13: SPLatticeDecodeLearner class.

```
PROCEDURE SPLatticeDecodeLearner::PickNextHypothesis()

DECLARE Sentence, CandGrammID;

/*    Sentence:         The current input sentence.
      CandGrammID:      A candidate grammar hypothesis. The learner
                        is entertaining the idea of making this
                        grammar the next hypothesis. */

Sentence ← Environment.GetAnInput();

IF Licensed(HypoGrammID, Sentence)
      THEN RETURN;
ENDIF;

CandGrammID ← GetRandomParseGrammarID(Sentence);

IF Lattice.IsAMemberOfSL(CandGrammID) THEN
      Lattice.Remove(HypoGrammID);
      HypoGrammID ← CandGrammID;
      RETURN;
ENDIF

ENDPROCEDURE;
```

Figure 14: SP Lattice Decode learner algorithm.

It should be noted that the underlying implementation of the learner may generate all the grammar IDs of a given sentence in parallel but it is not performing an actual parallel parse. From a linguistic perspective this does not constitute a parallel parse of the input sentence. The theoretical learner that the implementation is modeling does not have knowledge of any grammars generated in parallel. Choosing one grammar ID at random from the set of grammar IDs that can parse the input sentence is one way of implementing a theoretical serial parse. As long as the implemented learner does not use the existence of the other grammar IDs to help guide hypothesis selection in any way it is still a serial parse from the perspective of the theoretical learner being modeled.

2.3.4 Decode Favor Unmarked Learner

This SP Lattice Decode Favor Unmarked learner (Decode Favor Unmarked) variant is similar to the Decode learner. The only difference occurs during the decoding process. Instead of choosing treelets at random this learner will choose treelets according to their markedness. A marked value is always a superset of an unmarked value although some parameters do not control subset/superset relationships. This learner will always choose an unmarked value over a marked value. Choosing unmarked values will increase the chance of generating an SL grammar during the decoding process.

```
CLASS SPLatticeDecodeFavorUnmarkedLearner INHERITS SPLatticeLearner
      CLASSPROCEDURES
            Reset();
            PickNextHypothesis();
      ENDCLASSPROCEDURES

      CLASSVARIABLES
      ENDCLASSVARIABLES
ENDCLASS
```

Figure 15: SPLatticeDecodeFavorUnmarkedLearner class.

```
PROCEDURE SPLatticeDecodeFavorUnmarkedLearner::PickNextHypothesis()

DECLARE Sentence, CandGrammID;

/*    Sentence:         The current input sentence.
      CandGrammID:      A candidate grammar hypothesis. The learner
                        is entertaining the idea of making this
                        grammar the next hypothesis. */

Sentence ← Environment.GetAnInput();

IF Licensed(HypoGrammID, Sentence)
      THEN RETURN;
ENDIF;

CandGrammID ← PerformSerialParseChoosingUnmarked (Sentence);

IF Lattice.IsAMemberOfSL(CandGrammID) THEN
      Lattice.Remove(HypoGrammID);
      HypoGrammID ← CandGrammID;
      RETURN;
ENDIF

ENDPROCEDURE;
```

Figure 16: SP Lattice Decode Favor Unmarked learner algorithm.

2.3.5 Integrated Learner

This SP Lattice Integrated learner (Integrated learner) is similar to the Decode learner since decoding also drives hypothesis selection but it differs in the way it constructs a parse during decoding. Only treelets that are from SL grammars are available for patching into the parse tree. Constraining the pool of treelets in this manner guarantees that a parse generated during decoding will be parsable by an SL grammar.[18] However, there is now a possibility that no parse will be generated during the decoding process. The SL grammar treelet set is maintained by the supergrammar so there is no extra computational load. Learner efficiency should increase in comparison to the decode learners since it no longer relies on random chance to find a suitable parse, but constructs one (if possible).

```
CLASS SPLatticeIntegratedLearner INHERITS SPLatticeLearner
      CLASSPROCEDURES
            Reset();
            PickNextHypothesis();
      ENDCLASSPROCEDURES

      CLASSVARIABLES
      ENDCLASSVARIABLES
ENDCLASS
```

Figure 17: SPLatticeIntegratedLearner class.

[18] Note that this guarantee is implementation-dependant. Our implementation prohibits parameter treelet interaction between treelets drawn from the SL set. Such interactions could conspire in such a way as to result in a hypothesis language that is actually a superset of one or more SL languages. See further discussion in section 2.4.2.

```
PROCEDURE SPLatticeIntegratedLearner::PickNextHypothesis()

DECLARE Sentence, CandGrammID, SL, ParseGrammIDs, ParseGrammIDInSL;

/*    Sentence:         The current input sentence.
      CandGrammID:      A candidate grammar hypothesis. The learner
                        is entertaining the idea of making this
                        grammar the next hypothesis.
      SL:               Set of smallest language grammar IDs.
      ParseGrammIDs:    Set of grammar IDs that can parse the
                        current input sentence.
      ParseGrammIDInSL: Set of grammar IDs that can parse the
                        current input sentence that are also in SL.
*/

Sentence ← Environment.GetAnInput();

IF Licensed(HypoGrammID, Sentence)
      THEN RETURN;
ENDIF;

Lattice.Remove(HypoGrammID);

CandGrammID ← GetGrammarWithHighestRewardFromSL();

IF Licensed(CandGrammID, Sentence)
      HypoGrammID ← CandGrammID;
      RETURN;
ENDIF;

Lattice.Remove(CandGrammID);

SL ← Lattice.GetSmallestLanguages();

ParseGrammIDs ← GetParseGrammarIDs(Sentence);

ParseGrammIDInSL ← SetIntersect(SL, ParseGrammIDs);

IF ParseGrammIDInSL.Empty() THEN
      HypoGrammID ←
Lattice.PickRandomGrammarFromSmallestLanguages();
      RETURN;
ELSE
      HypoGrammID ← PickRandomGrammarFromSet(ParseGrammIDInSL);
ENDIF


ENDPROCEDURE;
```

Figure 18: Integrated learner algorithm.

2.3.6 Flashlight

The Flashlight is an add-on to other learners as opposed to a learner itself. The flashlight is an add-on that can be applied to most of the lattice learners. The flashlight stores a count, *c*, for each grammar; *c* tells how often subsets of that language have been hypothesized.[19] Whenever a new grammar is chosen (abiding by SP), all of the supersets of that grammar increment their *c* values by one. Whenever the learner needs to consider a new hypothesis, it will prioritize its choice by the highest *c* value among SL grammars. Choosing hypotheses in this manner is effective because when a language that is low in the LD lattice is disconfirmed, there is at least one sentence that is not in the disconfirmed language. There is a chance that this sentence may be a member of the set difference between the currently hypothesized language and one of its supersets. The probability of the current sentence being in the set difference does depend on the distribution of sentences in the language domain and the shape of the LD lattice. Because of the flashlight, the evidence the learner has encountered so far will direct the learner towards supersets of previous hypotheses rather than properly intersecting or disjoint languages. This heuristic encourages the learner to explore areas of the lattice that have worked well in the past, effectively implementing a hill-climbing strategy.

[19] We call it the flashlight because one can envision shining a flashlight up from the bottom of the lattice and all grammars that are illuminated have their counts incremented.

```
CLASS SPLatticeFlashlightLearner INHERITS SPLatticeLearner
      CLASSPROCEDURES
            Reset();
            SetFirstHypothesis();
            PickNextHypothesis();
            GetGrammarWithHighestRewardValueFromSL();
      ENDCLASSPROCEDURES

      CLASSVARIABLES
            DECLARE Flashlight;
            /* Flashlight:    Holds reward value counts for each
                               grammar in the lattice.
            */
      ENDCLASSVARIABLES
ENDCLASS
```

Figure 19: SPLatticeFlashlightLearner class.

```
PROCEDURE SPLatticeFlashlightLearner::PickNextHypothesis()

DECLARE Sentence, CandGrammID;

/*    Sentence:         The current input sentence.
      CandGrammID:      A candidate grammar hypothesis. The learner
                        is entertaining the idea of making this
                        grammar the next hypothesis. */

Sentence ← Environment.GetAnInput();

IF Licensed(HypoGrammID, Sentence)
      Flashlight.Reward(HypoGrammID);
      THEN RETURN;
ENDIF;

Lattice.Remove(HypoGrammID);

CandGrammID ← GetGrammarWithHighestRewardFromSL();

IF Licensed(CandGrammID, Sentence)
      HypoGrammID ← CandGrammID;
      RETURN;
ENDIF;

Lattice.Remove(CandGrammID);

CandGrammID ← GetGrammarWithHighestRewardFromSL();

ENDPROCEDURE;
```

Figure 20: SP Lattice Flashlight learner algorithm.

2.3.7 Largest Language Optimal Learner

The goal of implementing SP Lattice LL Optimal learner (LL Optimal learner) was to create an extremely efficient learner without regard for psychological plausibility, to serve as a benchmark learner. This learner is similar to a traditional search algorithm. The learner looks to prune the search space as much as possible with each input sentence. It essentially operates by pruning the search space from above and

searching from below. A largest language set (LL) is maintained by the learner. The largest language set contains all of the languages in the domain that do not have a superset. If the current hypothesis fails to parse the current input sentence then the learner looks to prune the lattice. It does this by parallel parsing the current input sentence with all the grammars in LL. Any grammars in LL that cannot parse the current input sentence are removed from the lattice. [20] Crucially, all subsets of those languages can also be removed since if the supersets cannot parse the current input sentence, then necessarily all their subsets also cannot. This allows the learner to remove considerably large chunks of the lattice, resulting in a similarly significant shrinking of the search space. Next the learner considers the bottom of the lattice and decodes the current input sentence. This decoding employs treelets that instantiate the grammars in the new SL set, resulting in a single grammar in SL that can parse the current input sentence (if one exists); this grammar becomes the current hypothesis. Otherwise, the learner removes all grammars in SL from the lattice and reconstructs SL. The new SL is checked against the set of grammars that can parse the current input sentence. The learner will keep removing grammars and reconstructing SL until it finds a grammar that can parse the current input sentence. When it does that grammar becomes the new hypothesis. Since the learner prunes the search space by performing a parallel parse of the input sentence it is not psychologically feasible. But as we show in section 2.4 Results and Discussion it serves well as our *optimal* learner.

[20] Note that serial decoding of the current input won't suffice since the learner needs to know which grammars *don't* license the current input and serial decoding gives only a single grammar that *can* license the input.

```
CLASS SPLatticeLLOptimalLearner INHERITS SPLatticeLearner
      CLASSPROCEDURES
            Reset();
            PickNextHypothesis();
      ENDCLASSPROCEDURES

      CLASSVARIABLES
      ENDCLASSVARIABLES
ENDCLASS
```

Figure 21: SPLatticeLLOptimalLearner class.

```
PROCEDURE SPLatticeLLOptimalLearner::PickNextHypothesis()

DECLARE Sentence, LL, SL;
DECLARE ParseGrammIDs, RemoveGrammIDs, CandGrammIDs;
/*    Sentence:         The current input sentence.
      LL:               Set of largest language grammar IDs.
      SL:               Set of smallest language grammar IDs.
      ParseGrammIDs:    Set of grammar IDs that can parse the
                        current input sentence.
      RemoveGrammIDs:   Set of grammar IDs to remove from the
                        lattice.
      CandGrammIDs:     Set of candidate grammar hypotheses. The
                        learner is entertaining the idea of making
                        one of these grammars the next hypothesis.
*/

Sentence ← Environment.GetAnInput();

IF Licensed(HypoGrammID, Sentence) THEN
      RETURN;
ENDIF;

LL ← Lattice.GetLargestLanguages();
ParseGrammIDs ← GetParseGrammarIDs(Sentence);
RemoveGrammIDs ← SetDifference(LL, ParseGrammIDs);

/* Add the previous hypothesis to the set of grammars to remove. */
RemoveGrammIDs.Add(HypoGrammID);

Lattice.RemoveIncludingAllDescendents(RemoveGrammIDs);

/* Recalculate this set since grammar IDs have been removed. */
ParseGrammIDs ← GetParseGrammarIDs(Sentence);

DO
      SL ← Lattice.GetSmallestLanguages();
      CandGrammIDs ← SetIntersect(SL, ParseGrammIDs);

      RemoveGrammIDs ← SetDifference(SL, ParseGrammIDs);

      IF CandGrammIDs.Size() = 1 THEN
            HypoGrammID ← CandGrammIDs[0];
      ELSE IF CandGrammIDs.Size() > 1 THEN
            HypoGrammID ← PickRandomGrammarFromSet(CandGrammIDs)
      ENDIF

      Lattice.RemoveIncludingAllDescendents(RemoveGrammIDs);

      /* Recalc ParseGrammIDs since grammar IDs were removed. */
      ParseGrammIDs ← GetParseGrammarIDs(Sentence);
WHILE CandGrammIDs.Size() = 0

ENDPROCEDURE;
```

Figure 22: LL Optimal learner algorithm.

2.3.8 Retrench Learner

The Retrench learner operates differently than the other SP Lattice learners. The Retrench learner is allowed to choose a grammar from anywhere in the LD lattice but it must *retrench* back down the lattice until it reaches a grammar that is safe with regards to superset errors. Retrenchment entails giving up the grammar that the learner just chose in the lattice in favor of a smaller grammar in the lattice if there is one (see Fodor & Sakas (2005) for a detailed discussion of retrenchment). Retrenching may cause the learner to give up the target language at this learning step, but it is necessary to ensure that the learner abides by SP. The Retrench learner has an advantage over the other SP lattice learners because it is not constrained to hypothesize grammars in SL but it is at a disadvantage because it is potentially susceptible to frequent undergeneralization problems due to the lack of memory; since SL grammars aren't deleted the depth of retrenchment (i.e., the amount of undergeneralization) is not reduced during the acquisition process. Undergeneralization problems are more likely to arise for the Retrench learner if the *domain ambiguity* is high. Domain ambiguity refers to the amount of overlap[21] between the sentences of each language in the domain. If there is a large amount of overlap between languages then domain ambiguity is high. Further discussion is presented in section 2.4.6. The starting point for retrenchment is determined by

[21] Throughout I use the term *overlap* to refer to the intersection of two sets. The sets may be either properly intersecting or in a subset-superset relationship.

decoding the current input sentence. Paths are chosen at random when choice points are encountered during decoding.

```
CLASS RetrenchLearner INHERITS Learner
      CLASSPROCEDURES
            Reset();
            SetFirstHypothesis();
            PickNextHypothesis();
            Retrench(GrammarID);
      ENDCLASSPROCEDURES

      CLASSVARIABLES
            DECLARE Lattice, CandGrammID;
            /*    Lattice:          Contains all the subset-
                                    superset relationships of the
                                    grammars.
                  CandGrammID:      A candidate grammar hypothesis.
                                    The learner is entertaining the
                                    idea of making this grammar the
                                    next hypothesis. */
      ENDCLASSVARIABLES
ENDCLASS
```

Figure 23: RetrenchLearner class.

```
PROCEDURE RetrenchLearner::PickNextHypothesis()

DECLARE Sentence;

/*    Sentence:         The current input sentence. */

Sentence ← Environment.GetAnInput();

IF Licensed(HypoGrammID, Sentence)
      Flashlight.Reward(HypoGrammID);
      THEN RETURN;
ENDIF;

ParseGrammIDs ← GetParseGrammarIDs(Sentence);

CandGrammID ← PickRandomGrammarFromSet(ParseGrammIDs);

/* Retrench will actually set the grammar hypothesis. */
Retrench(CandGrammID);

ENDPROCEDURE;
```

Figure 24: Retrench learner algorithm.

```
PROCEDURE RetrenchLearner::Retrench(RetrenchGrammarID, Sentence)

DECLARE CurrLatticeGrammID, SubsetGrammIDs, FoundRetrenchGrammar;

/*    CurrLatticeGrammID:     The current place in the lattice
                              where retrenching is taking place.
      SubsetGrammIDs:         Set of grammar IDs which are subsets
                              of the current place of retrenchment
                              in the lattice.
      SubsetGrammID:          Loop variable used while iterating
                              through the SubsetGrammIDs
                              collection.
      FoundRetrenchGrammar :  Boolean variable indicating whether
                              or not a subset grammar was found
                              that the learner can retrench to.*/

CurrLatticeGrammID ← RetrenchGrammarID;

DO
      SubsetGrammIDs ← Lattice.GetSubsetIDs(CurrLatticeGrammID);
      FoundRetrenchGrammar ← FALSE;

      FOREACH SubsetGrammID IN SubsetGrammIDs
            IF Licensed(Sentence, SubsetGrammID) THEN
                  CurrLatticeGrammID ← SubsetGrammID;
                  FoundRetrenchGrammar ← TRUE;
                  RETURN;
            ENDIF
      ENDFOREACH

WHILE FoundRetrenchGrammar = TRUE

HypoGrammID ← CurrLatticeGrammID;

ENDPROCEDURE;
```

Figure 25: Retrench procedure.

2.4 Results and Discussion

Each learner was run on all 3072 languages of the CoLAG domain. 100 trials per language were run. For each trial, learners were limited to a maximum of 10,000 input sentences. If the learner reached the maximum number of input sentences then the trial was considered a failure (i.e. the learner did not learn the language on that trial). The "Avg of 99% values" statistic is used to approximate the worst case performance of the learner. This value is calculated by taking the average of the 99th fastest trials for each of the 3072 languages in the CoLAG domain.

Learner	*Avg Sentences for Learned Trials*	*Avg of 99% values for Sentences for Learned Trials*	*Avg Parses for Learned Trials*	*Avg of 99% Values for Parses for Learned Trials*	*Pct Learned*
Lattice	858.69	1453.63	1588.99	2643.41	100
Decode	2482.54	6290.64	2482.54	6290.64	42
Decode Favor Unmarked	454.16	1686.99	454.16	1686.99	26
Integrated	140.43	286.24	140.43	286.24	100
LL Optimal	15.55	58.58	435.55	813.97	100
Lattice w/ Flashlight	900.88	968.28	1669.76	1744.68	100
Retrench	188.17	696.75	585.83	2134.15	95

Table 1: Lattice learner results on CoLAG domain.

2.4.1 LL Optimal Best For Sentences but Not Parses

As expected, the SP Lattice LL Optimal learner (LL Optimal learner) performs the best of all learners under study when measuring learner efficiency in terms of the number of sentences consumed, but performs notably worse than the most efficient psychologically feasible learner, the Integrated learner, in terms of the number of parses required. This learner is optimal in the sense that it is free to aggressively prune the search space and drastically reduce the number of languages that can be hypothesized. This pruning radically reduces the number of input sentences required but not the number of parses. The LL Optimal learner took three times as many parses as the Integrated learner took. When encountering an input sentence that the current hypothesis grammar can't parse, the LL Optimal learner must parse the current input sentence using the grammars of every language in the largest language set. This is necessary in order to determine which languages can (or cannot) be removed from the lattice. However, the LL Optimal learner efficiently decodes the input sentence using treelets from smallest language grammars, just as the Integrated learner does, the additional pruning phase employed by the LL Optimal learner increases the computational cost in terms of parses as compared to the Integrated learner.

2.4.2 Integrated Learner Performs Best

The SP Lattice Integrated learner (Integrated learner) is able to converge quickly on the target language because of the constrained treelet selection it uses during

decoding of the input sentence. It is guaranteed to choose a parse that corresponds to a language in the smallest language set, if one exists that can parse the current input. By contrast, the SP Lattice Decode learner (Decode learner) and SP Lattice Decode Favor Unmarked learner (Decode Favor Unmarked learner) also use the parser but they rely on random chance (in the case of the Decode learner) and a count of unmarked parameters (in the case of the Decode Favor Unmarked learner) to select a serial parse; there is no guarantee that the parse selected corresponds to a language that is currently in the SL set. If the selected serial parse does not correspond to a language in the SL set then the current hypothesis is retained and *no learning takes place*. There may in fact be many parses that do correspond to languages in the smallest language set that are overlooked by the Decode or Decode Favor Unmarked learners. Consequently, these learners miss learning opportunities due to incorrect parse selection (with respect to the SL set). Incorrect parse selection decreases the frequency of hypothesis changes, which in turn, decreases the number of times that a language can be removed from the search space. Hence memory for past grammars is underutilized.

Since the Integrated learner, on the other hand, is guaranteed to select a serial parse that corresponds to a language in the smallest language set (if one exists), the benefit of memory for past grammars is significantly increased. The Integrated learner will hypothesize different languages more often and consequently eliminate more languages from the search space.

Unfortunately, the guarantee of selecting a smallest language during a serial parse is difficult to establish within current linguistic theory. Our implementation constrains the learner in such a way that an SL grammar is chosen (if one exists). However, it is probably the case in the domain of natural languages that syntactic parameters can conspire in non-transparent ways (Fodor & Sakas, 2004). This might mean that two treelets drawn from the pool of SL set treelets, might in combination, guide the learner to entertain a non-SL set hypothesis. Obviously, this would be a fatal error if the non-SL set hypothesis was a superset of the target language. However, the positive results presented in this thesis of the efficiency of the Integrated learner recommend further linguistic investigation of whether or not these subset-superset parametric conspiracies could possibly be innately endowed, i.e., part of UG principles (see discussion in Fodor & Sakas 2005 of subset-superset parametric conspiracies).

2.4.3 Higher Parsing Priority Hinders The Decode Learner

The performance of the SP Lattice Decode learner (Decode learner) suffers because it prioritizes parsing over SP which is employed post-parsing to avoid potential superset errors. This learner operates by decoding the current input sentence in order to find a candidate hypothesis grammar. The candidate hypothesis grammar is checked to see if it is a member of the current SL set. If it is in the SL set then it becomes the current hypothesis grammar otherwise the previous hypothesis grammar is retained. Progress is made towards the target grammar only in the case that the learner chooses a

candidate hypothesis grammar that is a member of the current SL set. If the generated parse does not correspond to a grammar in the current SL set then, due to SP considerations, the learner must not change its current hypothesis. There may have been a suitable new hypothesis grammar in the SL set that could parse the current input which was missed by the random selection of treelets at choice-points encountered during decoding.

For this learner, target grammars that are not at the bottom of the lattice are harder to learn. This is the case because there are sentences in those target languages which aren't in SL set languages. During learning, the decoding process will often generate parses that correspond to non-SL set languages and consequently more input sentences will be discarded than if the target language were in the SL set.[22] To acquire a target grammar that is not in the SL set it is necessary for the learner to disconfirm all of the target grammar's subsets. Once all of the target grammar's subsets have been disconfirmed then the target will become a member of the SL set and it can be selected as a hypothesis. This learner can disconfirm grammars only after they are hypothesized. Once a grammar is disconfirmed the learner will use its memory for past grammars to ensure that the disconfirmed grammar is not selected again as a hypothesis grammar.

The distribution of sentences in the target language with respect to its subset and superset languages will determine how fast convergence will be for this learner. If

[22] Technically this isn't quite true. It could be the case that by sheer luck the learner always chooses treelets that correspond to languages in the SL set. Note that this situation reduces exactly to our Integrated Learner (see section 2.3.5 Integrated Learner).

there is not a large overlap between sentences of the target language (when the target is not a member of SL) and the sentences of languages in SL, then learning will be slow. The large percentage of sentences in non-SL set languages cause the learner to discard many input sentences and retain the previous grammar hypothesis. Inputs are wasted and learner efficiency is decreased. A large percentage of sentences in SL set languages will increase learner efficiency. Even if the learner chooses languages in the SL set other than the target it will at least be able to remove those grammars from the search space when they are disconfirmed. There will be far fewer times when the learner discards the current input sentence and just retains the current hypothesis.

2.4.4 CoLAG Is Unlearnable For The Decode Favor Unmarked Learner

The SP Lattice Decode Favor Unmarked Learner (Decode Favor Unmarked learner) is similar to the SP Lattice Decode learner (Decode learner) but is an attempt to improve performance by using a heuristic to increase attention to SP considerations. This learner will always choose the parse that corresponds to the most unmarked grammar. It does this by choosing an unmarked parameter value (treelet) when encountering a choice point during decoding. Exactly like the Decode learner, if the candidate hypothesis is in the SL set, it is chosen as the (new) current hypothesis otherwise the (previous) current hypothesis is retained.

Some languages in the CoLAG domain are unlearnable for the Decode Favor Unmarked learner. The unlearnable languages are a result of the unmarked grammar

count search heuristic. The unmarked grammar count search heuristic can sometimes cause the learner to get stuck in a local maximum. This occurs only under certain conditions, specifically, when the unmarked grammar search heuristic causes the parser to select candidate grammars that are incompatible with SP. This results in the learner not being able to change its hypothesis, since a grammar must necessarily be hypothesized in order to attain the target. The inability of the learner to hypothesize a given language results in that language, and all supersets of it, being unlearnable.

Any subset-superset pair of languages where the superset has a greater unmarked count than the subset will cause some languages in the domain to be unlearnable. The unlearnable languages will be the subset of that pair and all supersets of that subset. These languages are unlearnable as a result of the way the SP Lattice Decode Favor Unmarked Learner chooses a parse during decoding. The SP Lattice Decode Favor Unmarked Learner will always choose the parse that corresponds to the most unmarked grammar. If the superset has more unmarked values then the superset parse will always be chosen over the subset parse during decoding. The learner will now check to see if the superset parse corresponds to a language in the SL set. Since the superset is not currently in the SL set, the learner will just retain the previous grammar hypothesis. The superset grammar cannot become a hypothesis until it is a member of the SL set but it cannot be a member of the SL set until all of its subsets are removed from the lattice. In order for all of the subsets to be removed each subset must be hypothesized at some point during learning. This is where the problem arises. The subset with the lower unmarked count can never be hypothesized because the

learner will always favor the superset grammar as a candidate hypothesis due to its higher unmarked count. The superset grammar will always be favored as a candidate over the subset grammar but it cannot become a hypothesis until the subset grammar is removed. The subset cannot become a hypothesis because the superset grammar has a higher unmarked value count. This is a deadlock situation. Both the subset and the superset are unlearnable. In addition, all supersets of the problematic subset are also unlearnable. This is the case even if those supersets have lower unmarked value counts. Those supersets are unlearnable because they require the problematic subset to be removed from the lattice at some point and that will never happen.

The Decode Favor Unmarked learner makes use of memory for past grammar hypotheses but this does not help with the learnability of some languages in the domain. Current grammar hypotheses are removed from the lattice only when they fail to parse the current input sentence. This learner gives priority to the parser when selecting the next grammar hypothesis. If the parser chooses candidates that are not compatible with SP then the learner will retain the previous hypothesis because SP must be obeyed. The secondary status of SP creates problems for this learner just as it did for the Decode learner. The unmarked count constraint imposed on the parser only approximates SP considerations. The actual application of SP is done after parsing has been completed. Unfortunately, unmarked values for parameters don't necessarily correspond to subsets. In order for this learner to make progress towards a target that is not currently in the SL set, grammars in the SL set must be hypothesized. Hypothesizing grammars in the SL set becomes hard because the

grammars with the most unmarked values are being selected and those grammars may not be in the SL set at the moment. All parameters are used when counting marked and unmarked, not just subset/superset parameters. The non-subset parameters adversely affect the unmarked counts and cause some languages in the domain to be unlearnable.

There is a special case to the subset-superset pair unmarked count problem which is in fact learnable. If the subset of the pair starts out at the bottom of the lattice then both languages are learnable. They are learnable because there is a chance that the subset language can be chosen as the initial grammar hypothesis. Selection of the initial grammar hypothesis is done using a uniformly distributed random selection of the grammars at the bottom of the lattice. This random selection does not take the unmarked count into consideration so it is possible to hypothesize the subset of the offending subset-superset pair. If the subset is chosen then it will eventually be disconfirmed and stored in the memory for past grammars. It will no longer be in conflict with the superset grammar. Any problems related to this specific subset-superset pair would now be resolved.

Note that many of the problems for this learner would disappear if there were a transparent relationship between the parameter values and the languages they generate. I.e., if grammar markedness truly reflected the subset-superset relationships in the space of languages. This is stipulated a priori in many theoretical discussions of SP (see for example, the Subset Condition and Independence Principle of Manzini

and Wexler (1987) and Wexler and Manzini (1987) and the Simple Defaults Model of Fodor and Sakas (2005)) but clearly does not hold for the empirical work on the CoLAG domain presented in this thesis.

2.4.5 Impact of the Flashlight

The Flashlight is used as a type of memory for the success of past grammars. The goal of the flashlight is to point the learner towards the most successful grammars as it moves through the search space. The flashlight biases the learner towards hypothesizing languages that have most often contained an encountered input sentence; i.e., supersets of previously hypothesized languages. This keeps the learner focused on a specific area of the lattice that has proved successful at licensing inputs in the past.

Interestingly, results show that the SP Lattice Flashlight learner (Flashlight learner) performs about the same as the SP Lattice learner (Lattice learner) in terms of the *average* number of sentences and the *average* number of parses but it has greatly increased learner efficiency in terms of the *99% values* for those two metrics; from Table 1, without the flashlight Avg # sents: 858.69, 99% value: 1453.63 and with the flashlight Avg # sents: 900.88, 99% value: 968.28. Without the flashlight Avg # parses: 1588.99, 99% value: 2643.41 and with the flashlight Avg # parses: 1669.76, 99% value 1744.68. This implies that the variance across the learning times of the target languages in the domain has decreased dramatically.

Learner	*Subsets → Avg Sents*	*Avg Height → Avg Sents*	*Subsets → Avg Parses*	*Avg Height → Avg Parses*
Lattice	.80	.87	.82	.89
Flashlight	.06	.00	.07	.01

Table 2: Pearson r correlations for Lattice and Flashlight learners.

Analysis of the behavior of the Lattice learner with the flashlight attached reveals that, the flashlight offers the greatest benefit to target languages higher in the lattice. Target languages higher in the lattice are effectively deeper in the search space since (subset) languages below them need to be considered and eliminated first.[23] Table 2 illustrates the relationship between the location of a language in the lattice and the computational effort required by a given learner to acquire the target language. The numbers in Table 2 are Pearson r correlations which range from 1.0 to -1.0. A value of 1.0 means there is a perfect correlation between the sets of values being compared. For example, a high correlation between subsets and sentences would mean that an increase in the number of subsets would cause an increase in the number of sentences required to acquire the target language. A value of 0.0 means there is no correlation between the sets of values. A value of -1.0 means that as one value increases the other decreases and vice versa. The Lattice learner has a high correlation on all of the Pearson r comparisons that were measured (subsets to sentences, subsets to parses, avg height to sentences, avg height to parses). This means that languages that are

[23] Note that this is true since we are attaching the Flashlight heuristic to the SP-lattice learner. Other learners might or might not require entertaining lower languages in the lattice before ones higher in the lattice. We attached the Flashlight to this particular learner, since it is the most nondeterministic of the lattice learners and subsequently the best to determine the potential benefit of the Flashlight.

higher in the lattice are harder for it to learn. Importantly, the Flashlight learner shows no correlation between the location of the target language in the lattice and the computational effort required to acquire the target.

Why are the higher languages harder to learn for the Lattice learner? At a minimum, languages higher in the lattice have more subsets that have to be hypothesized and subsequently disconfirmed. All subsets of a language must be hypothesized and disconfirmed before the Lattice learner can hypothesize a given target language. When the Lattice learner selects a hypothesis language from the SL set there is no guarantee that the language being selected will be a subset of the target language. The Lattice learner is performing unnecessary work when it hypothesizes languages which are not subsets of the target language. Target languages higher in the lattice force the Lattice learner into making more random choices which increases the chances of selecting non-subsets of the target. An increased number of random selections causes an increased number of times that the learner will select garden path hypotheses, which will in turn increase the number of sentences necessary to acquire the target language. The flashlight learner decreases the amount of times that the learner selects non-subsets of the target language. It forces the learner to hypothesize languages which are subsets of the target and consequently reduces the amount of effort necessary for the learner to acquire the target language.

Results show that the Lattice learner and the Flashlight learner require about the same computational effort to acquire languages lower in the lattice. This makes sense

because acquiring languages lower in the lattice is more dependent on random chance. If a language is at the bottom of the lattice then all of the flashlight counts will be 0 and the Flashlight learner is reduced to selecting at random just as the Lattice learner does. For languages that are just above the bottom, say 1 level up, the flashlight will still not help much because all of the target language's subsets will have flashlight counts of 0. However, it will be easier to find the target language when it becomes a member of the SL set because it will have a flashlight count greater than 0.

The effectiveness of the flashlight depends on the percentage of subset-free triggers in the target language. A high percentage of subset-free triggers means that most sentences will increase only the target language's activation count[24]. A low percentage of subset-free triggers may or may not be bad. A small amount of subset-free triggers combined with numerous overlapping languages will mean that many languages will have their activation counts increased. A small amount of subset-free triggers combined with only a few overlapping languages will mean not as many languages with high activation counts and consequently the target language will be easier to find.

2.4.6 Retrench Learner Is Good But CoLAG Is Unlearnable

[24] The activation counts of all superset languages of the target language will also be increased but they are irrelevant because the only way those languages can become possible hypotheses is to disconfirm the target language and that can never happen with the SP Lattice Flashlight learner.

The Retrench learner only learned on 95% of the trials. This was due to the configuration of the lattice for certain languages and their subsets. For example, for language 3 in the CoLAG domain, all 100 trials failed. Language 3 has three subsets: Languages 1, 67 and 579. All of the sentences in language 3 are members of one of the subset languages. There are no subset-free triggers in language 3. The absence of subset-free triggers in language 3 means that language 3 is unlearnable using the Retrench learner. The Retrench learner will retrench down the lattice until there are no subset languages that can parse the current input sentence. For language 3 the Retrench learner will always retrench to one of its subsets because every sentence in language 3 is in one of language 3's subsets. There are no subset-free triggers that will allow the learner to hypothesize language 3. In general, any language that has no subset-free triggers is unlearnable using the Retrench learner. There is no way to escape the retrenchment problem that arises. The Retrench learner has no memory for past grammars so it cannot disconfirm the problematic subset grammars. The problematic grammars will always be allowed to become the current hypothesis. If memory for past grammars were used then this learner could escape the retrenchment problem. The learner would fall into the problematic subsets but those subsets would eventually be disconfirmed and removed from the lattice. As the subsets are removed sentences that were previously members of subsets now become subset-free triggers for the duration of the trial. It would now be possible to hypothesize the target grammar and therefore learn the language.

Language	*Total Sentences*	*Sentences in common with subsets*	*Subset-free Triggers*	*Pct of overlap*	*Pct of Learned Trials for Retrench Learner*
2642	312	300	12	96.15%	91%
2649	408	396	12	97.06%	77%
6754	279	278	1	99.64%	35%

Table 3: Retrench learner performance on selected languages from the CoLAG domain.

The percentage of subset-free triggers can also affect learner performance (see Table 3). In general, languages with higher percentages of subset-free triggers will be easier to learn. In the CoLAG domain some languages such as 2649 and 2642 are learned on some of their trials. Language 2649 has 77% of its trials learned whereas 2642 has 91% of its trials learned. The percentage of learned trials depends on the overlap percentage between sentences in the superset language and sentences in the union of the subset languages.

2.4.7 The Parser and SP Work Best in Tandem

Results showed that learners giving equal priority between parsing and SP considerations performed best. The equal priority between parsing and SP allowed learners to quickly attain hypotheses that were both relevant to the current input sentence and compliant with respect to SP constraints. Giving one or the other

priority seemed to decrease learner efficiency. Hypothesis selection becomes more of a hit or miss proposition when one is given priority over the other. For example, the Decode learners give the parser priority and as a result they must hope that the grammar associated with the constructed parse is compatible with SP constraints. If that grammar is not compatible with SP constraints then the previous hypothesis must be retained. Retaining the previous hypothesis means that nothing will have been learned from the current input sentence. More input sentences are needed in order to discover candidate hypotheses that are compatible with SP. This is a waste of computational resources. Giving SP priority can also decrease learner efficiency. The SP Lattice and SP Lattice Flashlight learners give SP priority. These learners will only select languages from the bottom of the lattice. Constraining hypotheses in this way is good for SP but it does not utilize information that is present in the current input sentence. The parser is employed simply to confirm or disconfirm hypotheses that were chosen strictly on the basis of SP considerations. Languages that are completely disjoint with respect to the target are free to be selected as the current grammar hypothesis. The side effects of selecting hypotheses that are not related to the input are not nearly as bad for the SP priority learners. These learners are able to utilize memory for past grammars and will not fall into the trap of selecting that hypothesis again. The effects of adding memory for past grammars are not masked like they are when the parser is given priority. When the parser settles on a parse that corresponds to a grammar that is not compliant with SP it must revert back to the previous hypothesis. Memory for past grammars cannot be utilized in this case because the previous grammar hypothesis was not eliminated. The benefits of

memory for past grammars are never seen using the Decode and Decode Favor Unmarked learners because SP compliant hypothesis grammars are slow to be selected or even impossible to be selected.

Memory for past grammars should increase learner efficiency as long as it is compatible with the learner. Compatible, in this sense, means that the learner is actually able to utilize the memory. In the case of the Decode learners, the memory was there but it was effectively unreachable. Poor parse selection forced the learner to retain the current hypothesis and consequently underutilize the memory. The SP Lattice learner did not perform best in terms of sentences or parses but it was still able to learn all of the languages in the domain. This high learnability was a result of the memory being accessible. The SP Lattice learner was able to continually change its hypothesis and as a result shrink the search space and eventually attain the target. Thus, as expected, memory for past grammars is effective as long the learner is of a kind that is able to use it.

3 Comparison of Partial and Total Ordering Learners

The learners that have been presented thus far all use a partial ordering (the lattice) to drive hypothesis selection and abide by the Subset Principle. Early research in language learning performed by Gold (1967) proposed using a total ordering of languages to drive hypothesis selection. Pinker (1979) has argued that the use of a total ordering to drive hypothesis selection is psychologically and computationally infeasible as a model of first language acquisition. We developed a partial ordering in order to keep the essential relationships given by a total enumeration while at the same time being psychologically and computationally feasible as a model of first language acquisition. This chapter investigates the efficiency of partial ordering learners compared to total ordering learners.

3.1 Why Use A Partial Ordering?

Although Gold's (1967) enumeration learner is the provably fastest learner given infinite classes of languages to be searched, in this chapter we set out to show that the partial ordering learners described in this thesis will be substantially faster than Gold's learner given a finite domain of languages. Gold's learner is only allowed to hypothesize languages in the order that they appear in the given enumeration. At any given point in the learning process the total ordering learner is only allowed to hypothesize the next language in the enumeration. With regards to SP, the only constraint for language learners in general is that they should hypothesize subsets of a given language before supersets of that language. The total ordering learner obeys SP but it cannot exploit the fact that other languages may not have a subset-superset relationship and that they are free to be selected with respect to SP. The partial ordering learner is similar to the total ordering learner in that it enforces SP but it is relieved of the burden of following a full enumeration of all the languages in the domain. The freedom that the partial ordering learner has with respect to hypothesis selection as compared to the total ordering learner will improve efficiency.

3.2 Total Ordering Learners

The total ordering learners that we have implemented operate in the same manner as Gold's learner with some slight modifications. Identification in the limit is not

invoked to determine when a learner has acquired the target language. Acquisition of the target language is defined to occur when the learner hypothesizes the target language or a language that is weakly equivalent to the target language. Identification by enumeration is used to determine hypothesis selection just as in Gold's learner. The enumeration given to the learners consists of all languages in the CoLAG domain. Gold proved that no learner using the identification by enumeration guessing rule is uniformly faster than any other learner using the identification by enumeration guessing rule so the order of the languages in our enumeration does not matter with respect to learner performance on the language domain as a whole. However, the enumeration will be constrained such that all subsets of a given language are guaranteed to appear before that given language in the enumeration. This ensures that the total ordering learner will abide by the Subset Principle. Languages which do not have subsets or supersets may appear anywhere in the enumeration.

3.3 Gold's Total Ordering Learner

What I will call *Gold's total ordering learner* is equipped with memory for all encountered input sentences. A candidate hypothesis is adopted if and only if it licenses all of the input sentences presented. The class definition for the learner is given in Figure 26 and the algorithm is detailed in Figure 27. The initial grammar hypothesis will be set to the first grammar in the enumeration. The learner gets a sentence from the input environment and checks to see if it is licensed by the current hypothesis. If it is then the current hypothesis will be retained otherwise a new

hypothesis will be chosen. When choosing a new hypothesis the learner will iterate through the enumeration until it finds a grammar that can license all of the input sentences that have been encountered.

```
CLASS TotalOrderingLearner INHERITS Learner
      CLASSPROCEDURES
            Reset();
            SetFirstHypothesis();
            PickNextHypothesis();

            LicensedAll(GrammarID);
            AddSentence(Sentence);
      ENDCLASSPROCEDURES

      CLASSVARIABLES
            DECLARE TotalOrdering;
            DECLARE SentenceVector;
            /* TotalOrdering: Enumeration containing all languages
                              in the CoLAG domain.
               SentenceVector:Collection of all input sentences
                              presented to the learner so far. */
      ENDCLASSVARIABLES
ENDCLASS
```

Figure 26: TotalOrderingLearner class.

```
PROCEDURE TotalOrderingLearner::PickNextHypothesis()

DECLARE Sentence, CandGrammID;

/*    Sentence:         The current input sentence.
      CandGrammID:      A candidate grammar hypothesis. The learner
                        is entertaining the idea of making this
                        grammar the next hypothesis. */

Sentence ← Environment.GetAnInput();
AddSentence(Sentence);

IF Licensed(HypoGrammID, Sentence)
      THEN RETURN;
ENDIF;

DO
      CandGrammID ← MoveToNextGrammarHypothesis();
WHILE (NOT LicensedAll(CandGrammID));

HypoGrammID ← CandGrammID;

ENDPROCEDURE;
```

Figure 27: Gold's Total Ordering learner algorithm.

3.4 Memoryless Total Ordering Learner

The memoryless version of the total ordering learner is the same as Gold's Total Ordering learner except that the memoryless version has no memory for past input sentences. The class definition for the learner is given in Figure 28 and the algorithm is detailed in Figure 29. The initial grammar hypothesis will be set to the first grammar in the enumeration. This learner will retain the current hypothesis as long as it can parse the current input sentence. If the current hypothesis cannot parse the current input sentence the learner will move through the enumeration until it finds a grammar that can parse it.

```
CLASS TotalOrderingMemorylessLearner INHERITS TotalOrderingLearner
      CLASSPROCEDURES
            Reset();
            SetFirstHypothesis();
            PickNextHypothesis();
      ENDCLASSPROCEDURES

      CLASSVARIABLES
      ENDCLASSVARIABLES
ENDCLASS
```

Figure 28: TotalOrderingMemorylessLearner class.

```
PROCEDURE TotalOrderingMemorylessLearner::PickNextHypothesis()

DECLARE Sentence, CandGrammID;

/*    Sentence:         The current input sentence.
      CandGrammID:      A candidate grammar hypothesis. The learner
                        is entertaining the idea of making this
                        grammar the next hypothesis. */

Sentence ← Environment.GetAnInput();

IF Licensed(HypoGrammID, Sentence)
      THEN RETURN;
ENDIF;

DO
      CandGrammID ← MoveToNextGrammarHypothesis();
WHILE (NOT Licensed(CandGrammID));

HypoGrammID ← CandGrammID;

ENDPROCEDURE;
```

Figure 29: Memoryless Total Ordering learner algorithm.

Although this version of the total ordering learner is more psychologically feasible as a model of human language acquisition than the Total Ordering learner (Figure 26 and Figure 27) since it does not endow the learner with the capacity to store an infinite number of input sentences, the number of parses per input sentence is unconstrained (up to the size of the domain) which is still too computationally demanding to be considered as a realistic model of human language learning.

3.5 Constrained Memoryless Total Ordering Learner

The constrained memoryless total ordering learner is the same as the memoryless total ordering learner except that it is limited to two parses per input sentence. The class definition for the learner is given in Figure 30 and the algorithm is detailed in Figure 31.

```
CLASS TotalOrderingConstrainedMemorylessLearner INHERITS
TotalOrderingMemorylessLearner
      CLASSPROCEDURES
            Reset();
            SetFirstHypothesis();
            PickNextHypothesis();
      ENDCLASSPROCEDURES

      CLASSVARIABLES
      ENDCLASSVARIABLES
ENDCLASS
```

Figure 30: TotalOrderingConstrainedMemorylessLearner class.

```
PROCEDURE
TotalOrderingConstrainedMemorylessLearner::PickNextHypothesis()

DECLARE Sentence, CandGrammID;

/*    Sentence:         The current input sentence.
      CandGrammID:      A candidate grammar hypothesis. The learner
                        is entertaining the idea of making this
                        grammar the next hypothesis. */

Sentence ← Environment.GetAnInput();

IF Licensed(HypoGrammID, Sentence)
      THEN RETURN;
ENDIF;

CandGrammID ← MoveToNextGrammarHypothesis();

IF Licensed(CandGrammID, Sentence)
      THEN HypoGrammID ← CandGrammID;
            RETURN;
ENDIF;

HypoGrammID ← MoveToNextGrammarHypothesis();

ENDPROCEDURE;
```

Figure 31: Constrained Memoryless Total Ordering learner algorithm.

The constrained memoryless version is the most psychologically feasible of the total ordering learners. It does not assume infinite memory for input sentences and realistically constrains the number of parses per input sentence; it is limited to a maximum of two parses per input sentence: One parse to test the current input sentence and another parse to test a candidate hypothesis. Most of the lattice learners were designed to be psychologically feasible with regards to memory and computation. Gold's learner in its original form is infeasible in both respects. All the lattice learners, except the LL-Optimal learner, respect the two parse per sentence

limit. By constraining Gold's learner similarly to the constraints we placed on our lattice learners, we are able to fairly compare the efficiency of both types of learners.

3.6 Discussion and Results

Learner	*Avg Sentences*	*Avg 99% for Sentences*	*Avg Parses*	*Avg 99% for Parses*
Gold's Total Ordering	15.63	52.00	1799.31	2635.65
Total Ordering Memoryless	22.26	62.38	1549.37	1589.49
Total Ordering Constrained Memoryless	902.80	964.36	1669.82	1731.67
SP Lattice	858.69	1453.63	1588.99	2643.41
LL Optimal	15.55	58.58	434.55	813.97
Integrated	140.43	286.24	140.43	286.24
Retrench	188.17	696.75	585.83	2134.15

Table 4: Comparision of Total Ordering and Lattice learners

3.6.1 Total Ordering Learner Inefficient In Terms of Parses

The Total Ordering learner (Gold's version) consumed almost exactly the same number of sentences on average as the SP Lattice LL Optimal learner (LL Optimal learner) before acquiring the target. As expected these learners yielded the best performance in terms of the number of input sentences. These learners needed

approximately 16 sentences on average to converge on the target. The memoryless version of the Total Ordering learner performed almost as well using approximately 22 sentences on average. Although performance of these learners is impressive, these results must be taken with a grain of salt because both the Total Ordering learner and the LL Optimal learner are psychologically infeasible due to the lack of constraints on the number of parses per input sentence. This psychological infeasibility is due to the excessive amount of parallel parsing[25] that these learners engage in. In the case of the LL Optimal learner, massive chunks of the search space are pruned off given the right input. Likewise, the Total Ordering learner can move through large sections of the enumeration on only one input sentence. This would be the case if an input sentence was unambiguous with respect to the target language and that target language was deep in the enumeration. The Total Ordering learner moves through multiple languages in the enumeration without requiring new input sentences. Note that the input sentence does not need to be completely unambiguous to move through large sections of the search space. For example, if the first input sentence was a member of 10 languages (assuming the CoLAG domain of 3072 languages) and those 10 languages were located somewhere in the final 1000 languages of the enumeration then the Total Ordering learner could move through the first 2072 languages of the search space using only that first input sentence. Of course, there is work going on even though unaccounted for in terms of the number of input sentences required by the learner. In order for the learner to adopt a new hypothesis in the enumeration, a parse is executed for *each* intervening grammar between that hypothesis and the

[25] Technically this parallelism could be replaced with a sequence of distinct serial parses, but either way the amount of computational cost is most probably beyond what is psychologically feasible.

current hypothesis. This implies that there is at least one parse performed for each and every language in the enumeration up to the target language. Now assume the target language is located at the end of the enumeration and the first input sentence is unambiguous with respect to that language. The Total Ordering learner will move through the whole enumeration on that single input sentence. The results of that trial in terms of number of input sentences consumed is not at all indicative of the amount of "work" that the learner performed. On the surface one input sentence is an extremely efficient result. In reality, what happened was the Total Ordering learner performed an exhaustive search of the language domain; the learner had to parse the input sentence using the grammars of every language in the domain. It did not *adopt* every language on the way to the target language, but it *considered* every one. In this example, although the cost in terms of number of inputs is 1, the cost in terms of the number of parses is 3072.

The number of sentences could also overestimate the amount of work being done by the learner. For example, suppose the target language is located near the beginning of the enumeration, say 20^{th} position. Also suppose that the stream of input sentences presented to the learner caused it to hypothesize each of the 19 languages located in front of the target language in the enumeration. That is, each input allowed the learner to move only one grammar forward. The cost in terms of sentences and parses is 20. Now compare the Total Ordering learner efficiency for this example against the previous example, which had the target language at the end of the enumeration. In terms of the number of input sentences, it took the learner much less time to acquire

the target language located at the end of the enumeration than it took the learner to acquire the language at the beginning of the enumeration. It appears that it is easier for the learner to acquire the language at the end of the enumeration. Of course this is misleading. The number of parses gives a better indication of the work done by the learner to acquire the target language. It took the total ordering learner 3072 parses to acquire the language at the end of the enumeration and only 20 to acquire the language at the beginning. This gives a more accurate representation of the effort put forth by the learner. We can now see that it took more work for the total ordering learner to acquire the language at the end of the enumeration than the language at the beginning.

The Total Ordering learner employs memory for past input sentences. During learning, the computational cost of using this memory is unaccounted for when using the number of input sentences as a metric. For each input sentence the Total Ordering learner parses the current input and all previous inputs encountered up until that point. For example, the computational cost to the Total Ordering learner in order to process the first ten input sentences is $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55$ parses. The number of parses is a good indication of the work performed by the learner due to memory for past sentences when faced with ambiguous input. Unlike the previous examples, suppose the total ordering learner is exposed to a string of *ambiguous* input sentences such as those depicted in Figure 32.

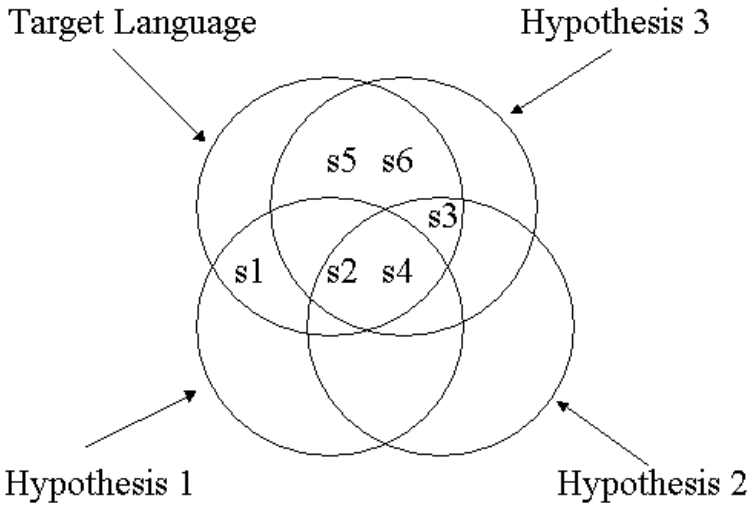

Figure 32: Language domain with ambiguous input sentences.

The input sentence memory store gives the Total Ordering learner a large advantage when exposed to this type of input. If each sentence were taken individually without memory, it might be hard for the learner to move through the enumeration towards the target language. The Total Ordering might hypothesize languages in the order shown. The sentences are ambiguous and they could easily be members of multiple non-target languages. The high ambiguity of these input sentences impedes movement through the enumeration toward the target language. Collecting input sentences and using the grammar associated with the current hypothesis to parse each member of the collection each time a new input sentence is encountered greatly increases the chances of disconfirming an invalid hypothesis. Now all input sentences must be members of the current hypothesis language in order for that language to be retained as the current hypothesis. The chances of disconfirming invalid hypotheses are greater because of the sentence memory store. For the example given in Figure

32, the Total Ordering learner would be forced to give up hypothesis 3 after the third input sentence was encountered. Importantly, the learner would also be compelled to hypothesize the target next because no other language contains all three input sentences. The input sentence memory store for the Gold learner increases learner efficiency in terms of the number of input sentences but it decreases learner efficiency in terms of the number of parses.

The Total Ordering learner performed the worst in terms of parses of all the learners being compared in this chapter. The input sentence memory store for the Gold learner turns out to be very expensive in terms of the number of parses. Each time a new input sentence is encountered that sentence and all of the sentences in the memory store must be parsed against the current hypothesis grammar. Consequently, the computational cost in terms of the number of parses will greatly increase as sentences are added to the memory store. Importantly, a major difficulty for the Total Ordering learner is the fact that it must necessarily parse every single grammar in the enumeration that is located before the target grammar. Even if a grammar is safe to skip over in terms of SP it must still be checked against all input sentences encountered to this point. These extra checks add to the computational cost. In contrast, the lattice learners have an advantage over the Total Ordering learners because they are only required to check languages that are supersets and/or subsets of candidate hypothesis languages. Unlike the Total Ordering learners, they are *not* required to check any languages that are not in a subset-superset relationship. This

freedom is what separates the lattice learners from the total ordering learners in terms of the number of parses.

3.6.2 Effects of Removing Total Ordering Input Sentence Memory Store

The Total Ordering Memoryless learner (Memoryless learner) is a modification of the Gold learner such that the input sentence memory store has been removed. The Memoryless learner does not test each input sentence encountered so far against a candidate hypothesis grammar and that results in less parses being needed for each individual input sentence. However, more sentences will probably be needed to disconfirm a false current hypothesis. The need for more input sentences will cause an increase in the number of parses and some of the efficiency gains realized by excessive parsing of each new grammar will be reduced. Results showed that the memoryless version outperforms Gold's version by approximately 250 parses on average. The additional parsing load caused by memory for past input sentences would seem to account for the difference in parsing efficiency of the two learners. The memory store is a double-edged sword. On the one hand the additional sentences make it easier to disconfirm false hypotheses but on the other they make it more computationally costly due to having to parse each individual input sentence. As the number of stored sentences increases, the cost of testing a new grammar hypothesis also increases. Adding new sentences to the memory store may have diminishing returns after a certain point. In practice, only a few sentences stored in memory may be necessary to greatly increase the ability of the learner to disconfirm false

hypotheses. If the size of the memory store were limited the learner may realize performance gains in terms of speedier disconfirmation of false grammar hypotheses without the burden of parsing every single input sentence for every single grammar hypothesis. Limiting the memory will mean less parses for a given input sentence but the learner will still benefit from extra input sentences during disconfirmation of a current hypothesis. Figure 32 (from section 3.6.1 Total Ordering Learner Inefficient In Terms of Parses) gives an example of a language domain where memory for only three input sentences would be necessary to see an increase in learner efficiency.

Performance of the Memoryless learner was only slightly worse than the Total Ordering learner in terms of the number of input sentences. The removal of the memory store only slightly affected performance. The Memoryless learner was still free to move through the enumeration as far as it could and so the number of input sentences remained very low. In domains with a higher level of ambiguity the difference in performance between these two learners would probably be more pronounced.

3.6.3 Constraining Total Ordering Learner Affects Performance

The Total Ordering Constrained Memoryless learner (Constrained Memoryless learner) was modified such that it is limited to two parses per input sentence. Constraining the Total Ordering learner in this manner creates a situation where the lattice learners and the total ordering learner can now be compared fairly. It would be

unjust to compare the learners when one is allowed unlimited parses for each input sentence and the other is not. Results showed that when using the number of input sentences as the measure of efficiency the Constrained Memoryless learner was slightly worse than the SP Lattice learner (Lattice learner). The parsing constraint forces the Constrained Memoryless learner to use more sentences in order to converge on the target. Gold's unconstrained version of the total ordering learner allows it to traverse the whole enumeration with only one input sentence since there is no constraint on the number of parses per input sentence.

Allowing the Total Ordering learner to move through the whole enumeration without regard for the number of parses is psychologically infeasible. Constraining the Total Ordering learner now makes its performance comparable to the Lattice learner's in terms of the number of input sentences. [26]

3.6.4 Partial Ordering More Efficient Than Total Ordering

The SP Lattice Integrated learner (Integrated learner) performed best of all the learners. The Integrated learner required only 140 parses on average as compared to the best total ordering learner that required 1549 parses on average. The Integrated learner was more than ten times faster than the best total ordering learner in terms of

[26] There was no constrained version of the Total Ordering learner (Gold's version with a memory store) because it would not have made sense since that learner must parse every input sentence in the memory store. The learner would have only been allowed to parse one sentence from the memory store and that would have had to be chosen at random. We could have removed the memory store completely and allowed only one parse for the current input sentence but that would have created an unfair comparison with the lattice learners which were allowed two parses per input sentence.

the number of parses. Importantly, the Integrated learner is a psychologically feasible model of first language acquisition and it was still able to outperform all of the total ordering learners regardless of their psychological feasibility. The ability of the Integrated learner to move about freely in the search space as compared to the total ordering learner allows it to skip portions of the search space that the total ordering learners cannot skip. The Integrated learner is able to outperform the total ordering learners in terms of the number of parses because of the flexibility it has to move around the search space and because of its ability to take full advantage of that flexibility. In contrast to the total ordering learners, the Integrated learner is able to focus on hypothesizing only the languages which are absolutely essential for it to disconfirm in order to ensure that SP is obeyed. It does not waste time hypothesizing languages that have absolutely nothing to do with the target grammar. This is the Achilles heel of the enumeration learners. The decoding process that the Integrated learner uses will only allow it to hypothesize grammars that are subsets of the target language (in the case that one exists in the SL set). Only the languages that are absolutely necessary to hypothesize are in fact hypothesized. The learner moves directly to the relevant areas of the search space. This is in contrast to the total ordering learners, which can spend time hypothesizing languages that are not subsets of the target and which may not even have any sentences in common with the target language. The total ordering learners are bound to examining the search space in a specific order for every single trial, no matter what evidence is presented to the learner. This rigid approach to navigating the search space will cause the computational cost in terms of the number of parses to have a lower bound of half the

size of the language domain on average. Utilization of a partial ordering as opposed to a total ordering frees the Integrated learner to search only the parts of the search space that are absolutely necessary to ensure that SP is being obeyed.

4 Effects of Language Domain Shape on Learning

The research discussed in this thesis thus far has focused on how different learning models operate in the CoLAG domain. The independent variable has been the type of learner used to model first language acquisition. We have investigated how each learner performs and what characteristics of each model contribute to their respective performances. This chapter will now turn to the question of how the shape of the language domain affects learner performance. We will approach this problem from two directions. First, we investigate learner performance across varying language domain shapes. Second, we will examine which learners performed best given a language domain shape. The language domains constructed for these simulations can be divided into two categories: subset language domains and properly intersecting language domains. The subset language domains contain languages that have subsets and/or supersets. The language domains in this category will vary according to their subset-superset relationships. The properly intersecting language domains do not have any pairs of languages that have subset-superset relationships. These language domains will vary according to the amount of unambiguous triggers that each language in the domain has. All languages within a given properly intersecting language domain will have the same amount of unambiguous triggers.

4.1 Why Examine Language Domain Shape?

Frequently, psychocomputational modeling of first language acquisition focuses on creating different models and examining how each of these models performs. One artificial language domain is posited and agents embodying one psycholinguistic acquisition theory or another attempt to converge on the languages in that domain. The existence of subset-superset relationships in a language domain adds more complexity to the problem of creating efficient models of first language acquisition. If subset-superset relationships do exist in the domain of natural human languages, then the question arises as to how best to shape the language domain for a psychocomputational modeling endeavor. What percentage of languages should have subsets? How many levels of subsets are there? Is the overall shape of the partial ordering of languages "taller" or "wider"? How much sentence overlap is there between languages in the domain? It is clear that learning performance can vary tremendously between different domains (e.g., Sakas 2000a, Sakas and Fodor 2001). The goal of the research presented in the chapter is to examine how language domain shape affects the learning performance of learners that obey the Subset Principle.

4.2 Tall Vs. Wide Lattices

The subset language domains we will be investigating can be grouped into two main categories, tall and wide. A tall lattice has most languages positioned on long vertical lines. If a tall lattice were viewed as a Venn diagram it would look like overlapping

onions each with many layers. A wide lattice has most languages positioned next to each other horizontally. There is very little deep nesting of languages in a wide lattice.

4.3 Domain Ambiguity Within The Language Domains

The ambiguity of sentences in the domain importantly contributes to its shape. The language domains were set up such that each language has one subset-free trigger or one unambiguous trigger. Languages at the top of the lattice will have at least one unambiguous trigger while languages that are not at the top of the lattice will have at least one subset-free trigger. The percentage of subset-free triggers and unambiguous triggers will vary by language and language domain shape. Within the subset language domains the percentage of subset-free triggers that a language has will depend on the number of subsets that the language has. If a language has a total of 49 subsets then it will have a total of 50 sentences resulting in only two percent of the sentences being subset-free triggers. On the other hand, if a language has one subset then it will have a total of two sentences giving a subset-free trigger percentage of 50 percent. Within the subset language domains only those languages at the top of any lattice will have unambiguous triggers. The properly intersecting language domains do not contain any languages in subset-superset relationships so languages in those domains will all have at least one unambiguous trigger. For the properly intersecting language domains, the percentage of unambiguous triggers will be dependent on the overlap of languages in the given domain.

4.4 Subset Language Domains

4.4.1 Description of Subset Language Domains

There are a total of 100 languages in each of the subset language domains. The subset language domains are constructed such that each language in each domain is in a subset-superset relationship with at least one other language in that domain. There are no properly intersecting languages in the subset language domains. The distribution of sentences is such that each language in each domain has one subset-free-trigger. Seven different language domains were created: 5-45-45-5, Skewed, 10 x 10, 50 x 2, 25 x 4, 4 x 25, and 2 x 50. The domains were constructed so that they reflect different types of shapes. Each domain is setup by row. For example, the 25 x 4 language domain has 25 rows and 4 columns. Each language in a given row is in a subset-superset relationship with each language in the row directly below it. In general, each row in a language domain is fully connected to the row directly below it.

4.4.1.1 Language Domain Shape - 5-45-45-5

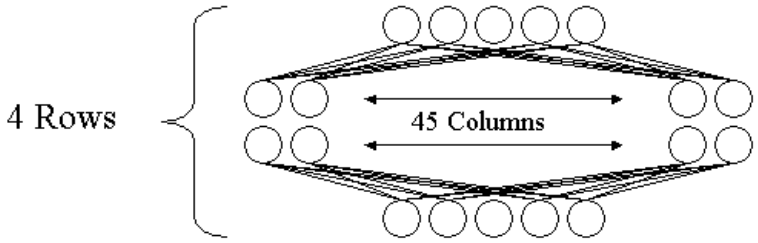

Figure 33: Language Domain Shape 5-45-45-5 Fully Connected

The 5-45-45-5 language domain has 4 rows with a variable number of columns and is a wide shape. All of the languages in a row are direct supersets of the languages of the adjacent row below it and all languages below that row are indirect subsets of it. Languages in the top row of this lattice will have 95 subsets each. There are only 5 languages in this domain that do not have any subsets.

4.4.1.2 Language Domain Shape - Skewed

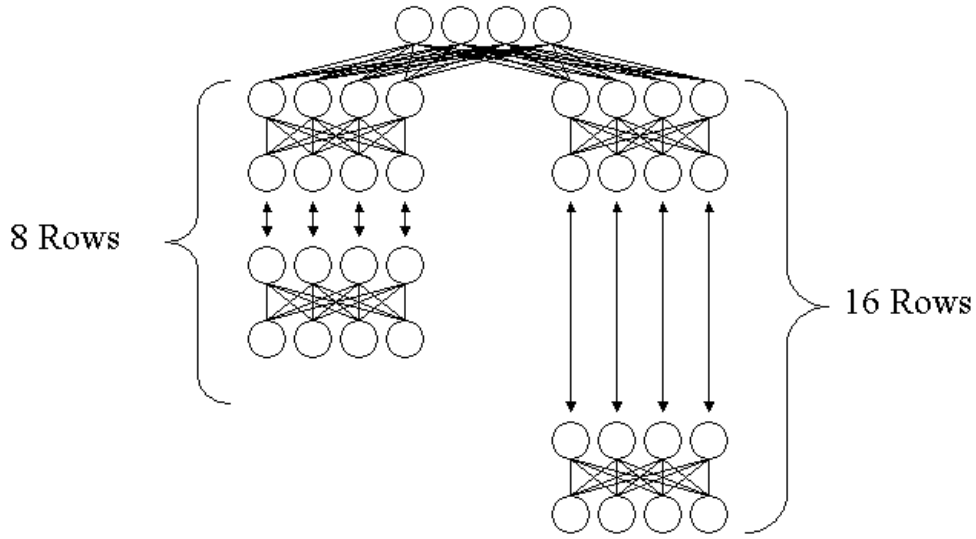

Figure 34: Language Domain Shape Skewed Fully Connected

The Skewed language domain was designed to be tall with two main branches. One branch has 8 rows and 4 columns and the other has 16 rows and 4 columns. Within a branch, all of the languages in a row are direct supersets of the languages of the adjacent row below it and all languages below that row are indirect subsets of it. The top row of the lattice has all languages fully connected to the top of each branch. Languages in the top row of this lattice will have 96 subsets each. There are only 8 languages in this domain that do not have any subsets.

4.4.1.3 Language Domain Shape – 10 x 10

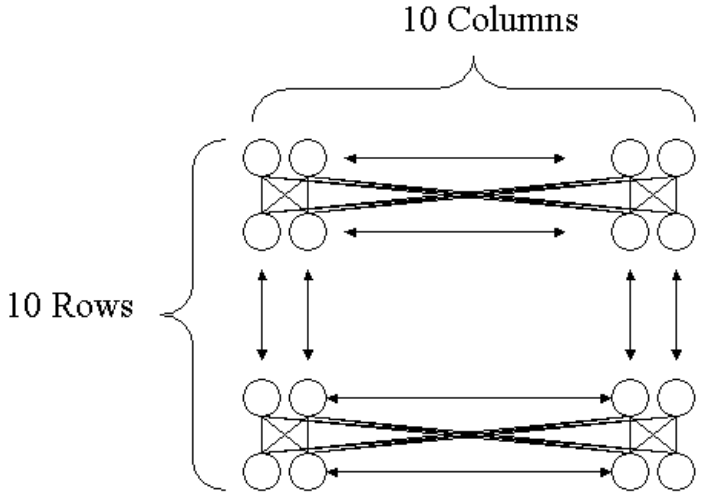

Figure 35: Language Domain Shape 10 x 10 Fully Connected

The 10 x 10 language domain has 10 rows and 10 columns and was designed to be a middle ground between tall vs. wide. All of the languages in a row, say row A, are direct supersets of the languages of the adjacent row below it, say row B. The languages in the rows below row B are indirectly subsets of the languages in row A. Languages in the top row of the lattice will have 90 subsets each. There are 10 languages in this domain which do not have any subsets.

4.4.1.4 Language Domain Shape – 50 x 2

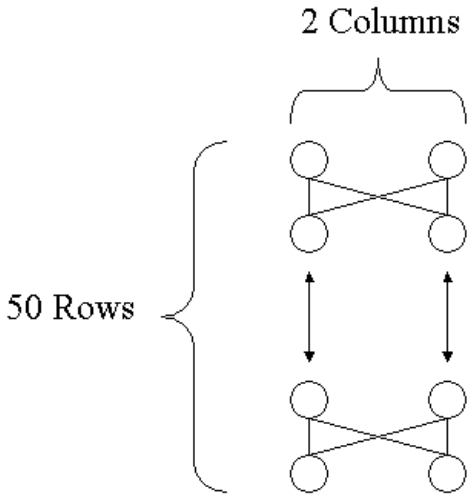

Figure 36: Language Domain Shape 50 x 2 Fully Connected

The 50 x 2 language domain has 50 rows and 2 columns and is an extremely tall shape. Each row in the language domain is fully connected to the row above it and below it. The languages in the top row each have 98 subsets.

4.4.1.5 Language Domain Shape – 25 x 4

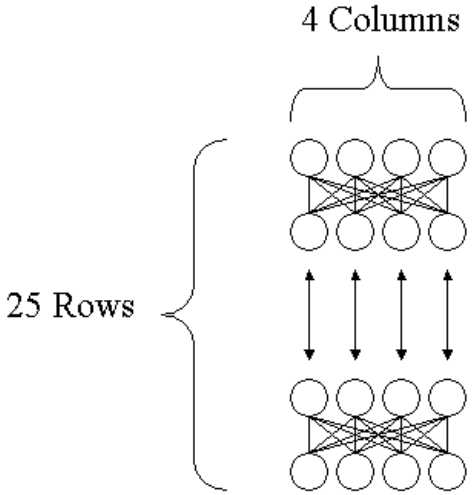

Figure 37: Language Domain Shape 25 x 4 Fully Connected

The 25 x 4 language domain has 25 rows and 4 columns and is a very tall shape. The goal of this design was to give the languages at the top of the lattice many layered languages to go through in order to acquire the target. Again, all of the languages in a row are direct supersets of the languages of the adjacent row below it and all languages below that row are indirect subsets of it. Languages in the top row of this lattice will have 96 subsets each. There are only 4 languages in this domain that do not have any subsets.

4.4.1.6 Language Domain Shape – 4 x 25

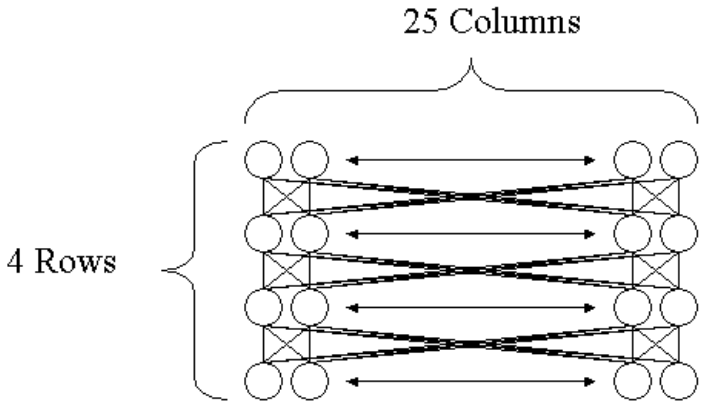

Figure 38: Language Domain Shape 4 x 25 Fully Connected

The 4 x 25 language domain has 4 rows and 25 columns and is a wide shape. All of the languages in a row are direct supersets of the languages of the adjacent row below it and direct subsets of the adjacent row above it. Languages in the top row of this lattice will have 75 subsets each.

4.4.1.7 Language Domain Shape – 2 x 50

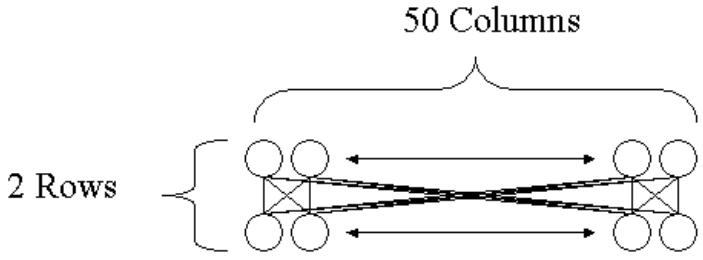

Figure 39: Language Domain Shape 2 x 50 Fully Connected

The 2 x 50 language domain has 2 rows and 50 columns and is a very wide shape. As before all of the languages in a row are direct supersets of the languages of the adjacent row below it and all languages below that row are indirect subsets of it. Languages in the top row of this lattice will have 50 subsets each. Half of the languages in this domain do not have any subsets at all.

4.4.2 Performance Across Subset Language Domains

4.4.2.1 SP Lattice Learner Performance Across Language Domains

Language Domain	*Avg Height*	*Avg Sentences*	*Avg 99% for Sentences*	*Avg Parses*	*Avg 99% for Parses*
5-45-45-5	1.50	310.74	703.01	336.22	736.85
Skewed	6.56	80.83	175.24	107.43	204.13
50x2	24.50	131.49	257.48	157.71	284.31
25x4	12.00	134.20	296.11	160.27	322.97
10x10	4.50	172.06	414.65	197.82	442.34
4x25	1.50	269.28	627.85	294.86	658.24
2x50	0.50	337.95	753.40	363.17	789.59

Table 5: SP Lattice learner results across language domains.

The relative performance of the SP Lattice learner on all five language domains was the same when using either sentences or parses as the measure of efficiency. The SP Lattice learner performed best for both sentences and parses on the Skewed language domain. The SP Lattice learner must eliminate all subsets of a given target language before it can hypothesize the target language. Most of the languages (96%) in the Skewed language domain are located in one of the two branches. The learner does not need to eliminate any languages from the other branch which reduces the number of languages that need to get eliminated on average for each language. For languages in either branch of the Skewed language domain the worst location for a target language would be at the top of the branch containing the 16 rows. Languages at the top of that branch have 60 subsets each (15 rows x 4 columns = 60 subsets). Contrast this with the languages at the top of the 25x4 language domain. Languages at the top of the 25x4 language domain have 96 subsets each (24 rows x 4 columns = 96 subsets). There are more languages that need to be eliminated on average in the 25x4 language domain as compared to the Skewed language domain. Each branch of the Skewed language domain is isolated from the other branch. This results in fewer subsets that need to be eliminated for languages located in either branch of the Skewed language domain.

4.4.2.2 Retrench Learner Results

Language Domain	*Avg Height*	*Avg Sentences*	*Avg 99% for Sentences*	*Avg Parses*	*Avg 99% for Parses*
5-45-45-5	1.50	30.51	111.36	796.63	2836.28
Skewed	6.56	28.55	99.04	108.81	328.36
50x2	24.50	49.22	177.50	131.61	335.20
25x4	12.00	48.77	175.99	131.14	372.90
10x10	4.50	46.42	172.94	228.80	745.87
4x25	1.50	38.88	139.46	638.04	2243.80
2x50	0.50	26.21	93.63	705.93	2495.01

Table 6: Retrench learner results across language domains.

For parses, the Retrench learner did best on taller as opposed to wider lattices (see Table 6). In order for the Retrench learner to safely hypothesize a given language it is necessary for it to hypothesize every immediate subset of that given language. For example, each language in the top row of the 2x50 language domain requires the learner to check all 50 subsets before the learner can safely hypothesize it. There is a lower bound of 50 parses for each language in the top row of the 2x50 language domain. Similarly, each language in the top row of the 5-45-45-5 language domain has a lower bound of 45 before each can be safely hypothesized. Retrench learner efficiency is worse for the 5-45-45-5 language domain because neither of the 45 wide

rows is at the bottom of the lattice. This causes a big problem for the five languages located in the top row of the lattice.

Location In Lattice	*Sents Avg*	*Parses Avg*
Top Row of 5	95.32	3013.31
Upper Row of 45	51.19	1382.72
Lower Row of 45	5.94	52.25
Bottom Row of 5	0.80	4.52

Table 7: Retrench learner results with languages grouped by row for the 5-45-45-5 language domain.

The results in Table 7 show that languages in the top row of the 5-45-45-5 language domain are very hard for the Retrench learner to acquire. The Retrench learner's efficiency suffers due to the large width of the two middle rows of the lattice. For example, suppose the target language is located in the top row of 5 languages, call it language *A*. Language *A* has 95 ambiguous sentences and only one unambiguous sentence. Of the 95 ambiguous sentences, 45 of them represent subset-free triggers for languages in the upper row of 45. Another 45 represent subset-free triggers for languages in the bottom row of 45. And finally, 5 of them represent subset-free triggers for languages in the bottom row of 5 languages. Now suppose the learner is presented with an input sentence that is a subset-free trigger for one of the languages in the upper row of 45 languages, call it language *B*. The learner will be required to parse the input sentence with all 45 of language *B's* direct subsets before it can safely hypothesize language B. None of the other grammars in the upper row of 45 succeed

in parsing the current input sentence and the learner will be free to hypothesize language *B*. The learner has just performed 45 parses on one input sentence and settled on hypothesizing language *B* and it is not even the target language. When trying to acquire language *A*, 47% (45 subset-free triggers from upper row of forty-five/96 total sentences in language A) of the sentences available to the learner will cause this scenario to happen. Another 52% (50 sentences in bottom two rows/96 total sentences) of the sentences, which are comprised of the subset-free triggers in the bottom two rows, will force the learner to hypothesize a language other than the target language. The computational cost required by the Retrench learner to hypothesize languages in one of the bottom two rows is small when compared to languages in the upper row of 45. The problem with hypothesizing languages in the bottom two rows is that they are not the target language (in this example) and the learner will be required to get another input sentence from the learning environment. The next input sentence will again have a 47% chance of being a subset-free trigger of language B or one of the other languages on the upper row of 45. The Retrench learner will spend a large amount of time in the bottom 95% of this particular lattice structure.

For sentences, the Retrench learner performed best on the 2x50 language domain. This is in contrast to the poor learner efficiency which resulted when parses were used as the metric. When the number of sentences is used to determine learner efficiency all parsing cost can be ignored. The large performance cost incurred as a result of parsing subsets of the target is not factored into the sentence performance

measure. There is also a better chance of selecting a subset-free trigger of the target language when using the 2x50 language domain as opposed to the 25x4 language domain. For the 2x50 language domain, the average percentage of subset-free triggers for all languages in the domain is 51 percent ((1/51 + 1/1) / 2). For the 25x4 language domain, the average percentage of subset-free triggers for all languages is only 7 percent ((1/97+1/93+…+1/4+1/1)/25). The Retrench learner requires subset-free triggers in order to hypothesize the target and it will encounter a higher percentage of subset-free triggers when learning in the 2x50 language domain as opposed to the 25x4 language domain.

The Retrench learner performs dramatically better on taller lattices when using parses as the metric and better on wider lattices when using sentences as the metric. The cost of parsing direct subsets in a wide lattice seems to outweigh the cost of deep retrenchment that may occur in tall lattices.

4.4.2.3 Total Ordering Learner Results

Language Domain	*Avg Height*	*Avg Sentences*	*Avg 99% for Sentences*	*Avg Parses*	*Avg 99% for Parses*
5-45-45-5	1.50	28.85	106.12	637.79	2261.31
Skewed	6.56	24.22	88.27	161.02	382.52
50x2	24.50	41.67	141.17	263.85	599.02
25x4	12.00	42.14	154.65	280.96	673.91
10x10	4.50	41.40	145.52	354.43	1021.84
4x25	1.50	36.46	134.51	549.37	1868.82
2x50	0.50	25.14	88.29	676.50	2354.64

Table 8: Total Order learner results across language domains.

The Total Ordering learner also performed best on the Skewed language domain. The reason for this may be because the Skewed language domain has a lower average number of subsets compared to most of the other language domains. The main problem for the Total Ordering learner may be coming from its inability to disconfirm incorrect hypotheses. Figure 40 shows an example of a target language that is hard for the Total Ordering learner to acquire.

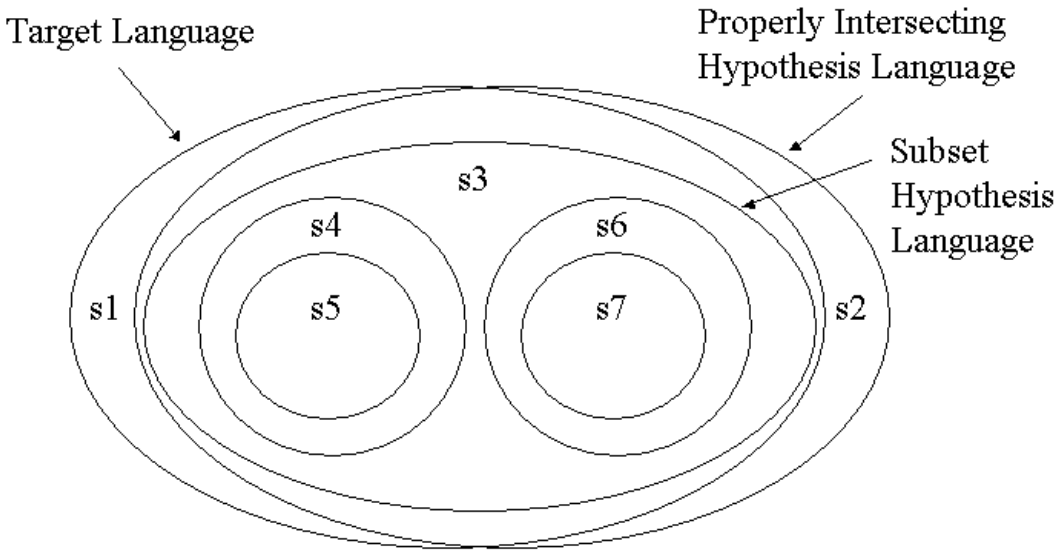

Figure 40: Target language that is hard for the Total Ordering learner to acquire.

In this example, the target language and two possible hypothesis languages are shown. Both the subset hypothesis and the properly intersecting hypothesis will be hard for the Total Ordering learner to give up because they both overlap with the target language to a large degree. The properly intersecting languages are a concern for the Total Ordering learner because those languages have to be considered as hypotheses when they appear before the target in the enumeration. In contrast, the SP Lattice learner is not required to consider the properly intersecting languages, therefore it is not as affected by them as the Total Ordering learner is.

Wider language domains with many subset-superset relationships will cause this problem to occur for the Total Ordering more than taller language domains. The 2x50 language domain is particularly hard for the Total Ordering learner because the

languages in the top row are all properly intersecting and have a large amount of overlap between them.

4.4.2.4 Total Ordering Constrained Memoryless Learner Results

Language Domain	*Avg Height*	*Avg Sentences*	*Avg 99% for Sentences*	*Avg Parses*	*Avg 99% for Parses*
5-45-45-5	1.50	313.53	502.08	339.02	527.59
Skewed	6.56	77.01	145.70	102.82	171.68
50x2	24.50	132.66	255.07	158.89	281.58
25x4	12.00	134.64	255.33	160.71	281.68
10x10	4.50	172.36	339.15	198.12	365.06
4x25	1.50	270.48	475.19	296.08	500.72
2x50	0.50	337.01	521.37	362.22	546.59

Table 9: Total Ordering Constrained Memoryless learner results across language domains.

The performance of the Total Ordering Constrained Memoryless learner was similar to the other learners. Again, the Skewed language domain was easiest to learn while the 2x50 language domain was the hardest. This learner encountered problems similar to those seen in the Total Ordering learner.

4.4.2.5 SP Lattice Flashlight Learner Results

Language Domain	*Avg Height*	*Avg Sentences*	*Avg 99% for Sentences*	*Avg Parses*	*Avg 99% for Parses*
5-45-45-5	1.50	311.01	519.95	336.50	546.21
Skewed	6.56	76.23	161.09	101.86	188.21
50x2	24.50	131.54	258.93	157.54	285.61
25x4	12.00	134.23	295.90	160.30	322.69
10x10	4.50	171.62	373.97	197.38	400.86
4x25	1.50	268.46	499.82	294.07	526.75
2x50	0.50	336.50	541.61	361.71	568.45

Table 10: SP Lattice Flashlight learner results across language domains.

The SP Lattice Flashlight learner also performed best on the Skewed language domain and worst on the 2x50 language domain. The results were similar to the other learners.

4.4.2.6 Discussion

In terms of the number of parses, all of the learners had almost the same relative order of performance for all of the language domains (see Table 11).

	SP Lattice	*Retrench*	*Total Ordering*	*Total Ordering Constrained*	*Flashlight*
1	Skewed	Skewed	Skewed	Skewed	Skewed
2	50x2	25x4	50x2	50x2	50x2
3	25x4	50x2	25x4	25x4	25x4
4	10x10	10x10	10x10	10x10	10x10
5	4x25	4x25	4x25	4x25	4x25
6	5,45,45,45	2x50	5,45,45,45	5,45,45,45	5,45,45,45
7	2x50	5,45,45,45	2x50	2x50	2x50

Table 11: Relative performance of learners on language domains in terms of the number of parses (fastest (1) to slowest (7)).

Table 11 illustrates which language domains were easiest and hardest to learn for each learner under investigation. For example, the SP Lattice learner performed best on the Skewed language domain and worst on the 2x50 language domain. Overall, the very wide language domains were the hardest to learn while the taller language domains were easier. Why might this be so? On the surface it would seem that language domains with a lower average number of subsets per language would be easier to learn but interestingly this was not the case (see Table 12).

Language Domain Shape	*Average Number of Subsets Per Language*
50x2	49.0
25x4	48.0
10x10	45.0
4x25	37.5
5-45-45-5	30.0
Skewed	28.0
2x50	25.0

Table 12: Average number of subsets per language by language domain.

The Skewed language domain was the easiest to learn while the 2x50 language domain was the hardest yet they both had roughly the same average number of subsets per language. The shape of the language domain was the main reason for the difference in performance. Why is the 2x50 language domain so hard to learn for all of the learners? For this language domain there is a great disparity in the learning times for languages in the top row as compared to languages in the bottom row across all of the learners (see Table 13).

	SP Lattice	*Retrench*	*Total Ordering*	*Total Ordering Constrained*	*Flashlight*
Top Row	701.56	1385.21	1327.52	699.43	698.36
Bottom Row	24.78	26.65	25.48	25.00	25.06

Table 13: Average number of parses for languages in either the top or bottom row of the 2x50 language domain.

The wide shape of this language domain caused problems for all of the learners under investigation. For languages in the top row, there are many subsets that need to be either hypothesized or at least evaluated as well as lots of properly intersecting languages that can be hard to disconfirm. These factors impacted all of the learners although the degree to which either of these factors impacted a given learner varied.

4.5 Properly Intersecting Language Domains

This section will examine how domain ambiguity affects learner performance. The language domains under investigation in this section do not have any languages in subset-superset relationships. High ambiguity language domains contain very few unambiguous sentences. Figure 41 gives an example of a highly ambiguous language domain.

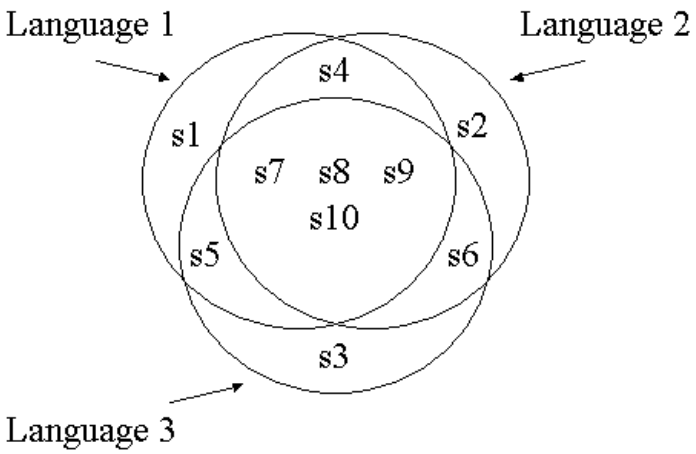

Figure 41: Language domain with high domain ambiguity.

Most of the sentences of each language in Figure 41 belong to at least one other language in the domain. Sentences S7 through S10 belong to every language of the domain and provide very little information that the learner can use to determine the target language. Sentences S4 through S5 are not members of every language but they are still ambiguous. They may contain some information that the learner can use. Sentences S1 through S3 are unambiguous with respect to the languages they belong to and should provide the learner with the most information. Figure 42 gives an example of a language domain with low domain ambiguity.

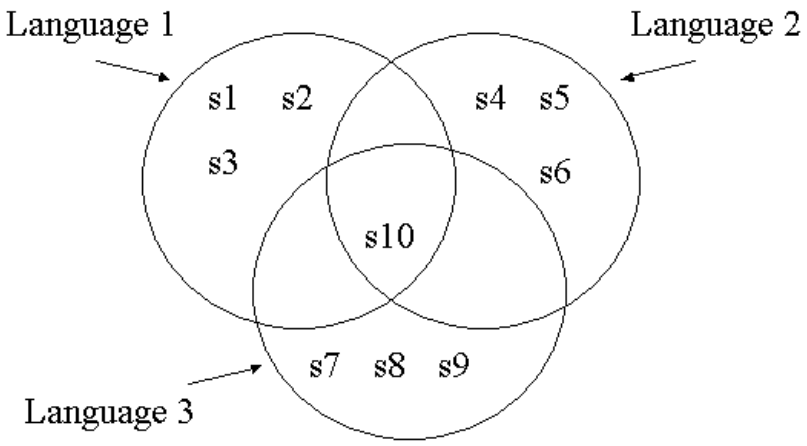

Figure 42: Language domain with low domain ambiguity.

Most of the sentences in this language domain are unambiguous. Only sentence S10 belongs to more than one language. In this section several learners will be run on language domains containing varying amounts of domain ambiguity in order to investigate the effects of domain ambiguity on learner efficiency.

4.5.1 Description of Properly Intersecting Language Domains

Each of the properly intersecting language domains contains 100 languages. These language domains will vary according to the average percentage of unambiguous sentences for languages in the domain. For example, the average percentage of unambiguous sentences for the languages depicted in Figure 41 is 14%. Each language in the domain contains seven sentences, only one of which is unambiguous

(1/7 = .14 = 14%). The average percentage of unambiguous sentences for the language domain given in Figure 42 is 75% (3/4 = .75 = 75%). Each language in the domain contains four sentences, three of which are unambiguous. A language domain with a high average percentage of unambiguous sentences means that most of the sentences in a given language belong only to that language. Language domains with low average percentages of unambiguous sentences contain languages with many sentences that are members of more than one language.

4.5.2 Learner Performance On Properly Intersecting Language Domains

In Table **14** through Table **18**, we give results for each learner on each of the properly intersecting language domains.

Pct Of Unambiguous Sentences	*Avg Sentences*	*Avg 99% for Sentences*	*Avg Parses*	*Avg 99% for Parses*
4%	160.47	474.07	190.00	519.55
12%	39.52	86.91	65.60	135.52
34%	27.33	54.17	52.79	103.07
51%	25.67	50.21	50.74	98.92
84%	25.06	48.70	50.06	97.28
99%	25.16	48.49	50.32	96.98

Table 14: SP Lattice learner results across properly intersecting language domains.

Pct Of Unambiguous Sentences	*Avg Sentences*	*Avg 99% for Sentences*	*Avg Parses*	*Avg 99% for Parses*
4%	87.45	405.51	96.98	428.09
12%	9.84	40.20	13.13	48.26
34%	2.62	8.93	4.51	13.55
51%	1.69	4.90	3.15	8.13
84%	1.09	2.13	2.17	4.12
99%	1.00	1.00	1.98	2.00

Table 15: Retrench learner results across properly intersecting language domains.

Pct Of Unambiguous Sentences	*Avg Sentences*	*Avg 99% for Sentences*	*Avg Parses*	*Avg 99% for Parses*
4%	15.48	45.68	126.27	224.00
12%	4.65	12.44	67.17	95.46
34%	2.03	4.80	54.19	63.36
51%	1.48	2.94	51.91	56.13
84%	1.08	1.62	50.69	51.90
99%	1.00	1.00	50.49	50.49

Table 16: Total Ordering learner results across properly intersecting language domains.

Pct Of Unambiguous Sentences	*Avg Sentences*	*Avg 99% for Sentences*	*Avg Parses*	*Avg 99% for Parses*
4%	78.59	121.20	108.18	151.12
12%	36.05	47.04	62.37	73.59
34%	27.15	30.83	52.59	56.67
51%	25.74	27.57	50.93	53.06
84%	25.09	25.57	50.11	50.73
99%	25.00	25.00	50.00	50.00

Table 17: Total Ordering Constrained Memoryless learner results across properly intersecting language domains.

Pct Of Unambiguous Sentences	*Avg Sentences*	*Avg 99% for Sentences*	*Avg Parses*	*Avg 99% for Parses*
4%	83.81	159.65	113.26	190.59
12%	36.28	50.41	62.60	78.54
34%	27.12	32.37	52.51	60.14
51%	25.72	29.34	50.89	57.03
84%	25.11	27.76	50.15	55.34
99%	24.98	27.53	49.97	55.06

Table 18: SP Lattice Flashlight learner results across properly intersecting language domains.

For every learner, learner efficiency was better for language domains with a higher percentage of unambiguous sentences and worse for language domains with a lower percentage of unambiguous sentences. Incremental changes in unambiguous sentence percentage affect learner efficiency more for low percentages. For each learner, the largest difference in computational efficiency for all metrics occurred between the language domains containing 4% unambiguous sentences and 12% unambiguous sentences. Increasing the percentage of unambiguous sentences past 34% did not significantly impact learner performance for the learners under investigation. It would seem that learners greatly benefit from increases in the percentage of unambiguous sentences to a point. There are diminishing returns for increases in the average number of unambiguous sentences past 34%. A percentage of 34% means that at least one-third of the sentences encountered by the learner will be unambiguous. The learner will quickly disconfirm incorrect hypotheses due to the unambiguous input and quickly move through the search space towards the target language.

4.6 Discussion

This preliminary investigation into the effects of language domain shape on SP-learner efficiency seems to indicate that the shape of the language domain does have an effect. Why does it have an effect? In some aspects, the shape of the language domain affects domain ambiguity. If a language has many subsets then the language necessarily overlaps those subset languages resulting in domain ambiguity. We have shown that domain ambiguity affects SP-learner performance therefore it is logical to

conclude that shape will affect learner performance as well. More importantly, our learners are required to abide by the Subset Principle. Faithful application of the Subset Principle is what makes language domain shape an issue. The Subset Principle forces learners to make safe hypothesis selections with respect to superset errors. For example, the Retrench learner (see section 2.3.8 Retrench Learner) must check all of a language's direct subsets before it can safely hypothesize that language. If a language has many direct subsets then the Retrench learner will necessarily be doing a lot of parsing in the name of the Subset Principle. On the other hand, if that language has many subsets but most of those subsets are indirect then it will not have to do as much parsing to hypothesize that language. In Figure 43, language A is hard for the Retrench learner to hypothesize because it must check all 6 subsets before it can safely hypothesize it.

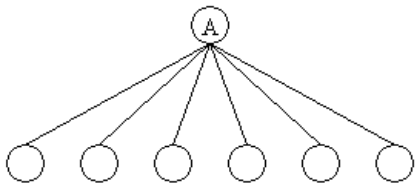

Figure 43: Language shape that is hard for the Retrench learner.

In Figure 44, language A is easier for the Retrench learner to hypothesize because it does not need to check all 6 subsets. It only has to check its two direct subsets.

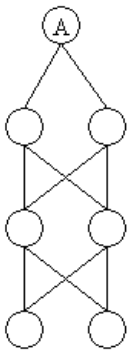

Figure 44: Language shape that is easy for the Retrench learner.

Domain ambiguity gives an indication of how hard it may be to learn a language domain but it does not give the full picture. There are subset-superset dependencies that must be considered in order to abide by SP. The shape of the language domain will directly affect the computational cost of SP compliance. Depending on the learner, the effects of language domain shape could be offset somewhat by the distribution of subset-free triggers and unambiguous triggers in the language domain although the shape will force lower bounds on computational cost no matter what the distribution of subset-free triggers and unambiguous triggers happens to be.

Overall, the current research indicates that language domains containing more breadth are harder to learn than language domains containing more depth. When abiding by the Subset Principle the learner is better off navigating a language domain containing vertical chains of subset/supersets as opposed to horizontal ones. It should be noted that this result does not take into account the connectedness of the language domain.

The current research was done using language domains that have a high degree of connectivity.

This research is only a first step in examining the effects of language domain shape on learner efficiency. There are many variables related to language domain shape that should be manipulated and evaluated in order to provide a more complete analysis. For example, the shapes of the language domains presently under investigation contain a high degree of connectedness. Each language in each row is a superset of each language in the row below it except in the case of the Skewed language domain. Interestingly, the Skewed language domain was the easiest to learn of all the language domains. The Skewed language domain forces a natural break in the domain, reducing the overall connectivity. Language domains that vary according to this type of connectivity should be investigated. Another variable that should be analyzed is domain ambiguity within the subset-superset language domains. Language domains should be created that vary the amounts of subset-free triggers and unambiguous triggers for languages in the domain. In addition, larger language domains as well as language domains that are linguistically plausible should be investigated with regards to language domain shape.

5 Conclusions, Implications and Future Research

This thesis provides a study of computational models of first language acquisition that implement the Subset Principle (SP). Importantly, a central concern throughout was the effect of constraining the learning algorithms to the extent that they make use of a psychologically viable amount of computational resources. The research presents:

- An empirical analysis of eight SP-compliant simulated language learners designed to exploit the partial ordering of subset inclusion in a psychologically feasible abstract language domain (the CoLAG domain).

- A comparison of these learners against learners that use a total ordering (full-enumeration) of all the languages in the domain in which subsets are posited before supersets to guide learning.

- An analysis of how the arrangement of the partial ordering of different language domains, what we call the domain's "shape", affects the performance of SP-compliant simulated language learners.

5.1 Summary of Findings

The central results of this thesis are the following:

- The hypothesis selection strategy of giving equal priority between the parser and SP-compliance works best. This is because this strategy directly focuses its attention to both viable hypotheses in terms of SP, *and* viable hypotheses in terms of the hypotheses' ability to license the input the learner encounters. The other SP-compliant learners examined in this thesis prioritized search strategies to either identify safe subset hypotheses or to find compatible hypotheses. This prioritization comes at a substantial cost.[27] Unfortunately this learner gets "for free" knowledge of which combinations of parameter values instantiate a guaranteed smallest subset language. However, linguistic research on syntactic parameters has shown that parameter values often interact in complex ways. So it wouldn't be trivial for the learner to come by this knowledge. Given parameter interaction, in effect, the learner would have to calculate the smallest language status of each of the 2^n languages in the domain (where n is the number of parameters). Still, the strong performance of this strategy would suggest that perhaps it would be worthwhile to develop a linguistically viable parametric system where subset parameter values are known or easily accessible.[28]

- Partial ordering learners outperformed total ordering learners in terms of the number of parses. The thesis makes an argument for using number of parses rather than number of sentences as a metric to measure workload since it

[27] Though other work described in Chapter 3 and briefly below indicates that the cost overall is not as extensive as the cost incurred by using a total-enumeration of a lattice.

[28] Note that the Decode Favor Unmarked learner (2.3.4 Decode Favor Unmarked Learner) would reduce to this learner if the linguistics of the CoLAG domain matched this description.

takes into account the computational effort expended during parsing an input sentence or sentences. The Total Ordering learner must parse every input sentence in the memory store each time it tests the next candidate grammar in the total ordering (enumeration). Since the learners must test each and every language up to the target language in the enumeration against its memory store, the parsing workload is tremendous for the Total Ordering learners when the target is more than a few grammars in from the beginning of the enumeration. Parsing workload is an element of the Gold paradigm that is largely overlooked. This result solidifies points made in discussion of the issue by Fodor and Sakas (2005) (cf. Pinker, 1979) and indicates that a partial ordering of subset-superset relationships is a beneficial advance over identification by enumeration.

- As a preliminary result, language domains which have large breadth are harder to learn than language domains which have large depth. This result is a consequence of SP implementation in the various learners under study. By definition, a language domain with large breadth has more sister nodes at each level in the lattice compared with a domain with large depth which has few nodes at each level. In general, to implement SP, i.e., to insure that the target is safe to adopt, at the very least all direct subset languages of the target must be considered in some fashion or another. For domains with large breadth this is computationally expensive beyond what is required to go deep into a narrow tall lattice. It is probably the case that the domain of natural languages

is wider than it is tall; in fact this is the case with the CoLAG domain. This would lead to the conjecture that psychologically plausible learners must be adept at handling domains where supersets contain a multitude of sister subsets.

5.2 Future Research

There are many areas of research that can be pursued with respect to the current research. In this thesis we added memory for *past grammars* to simulated language learners. It would be interesting to analyze the effects of adding an *input sentence* memory store, such as the one used by the Total Ordering learner, to our SP-compliant simulated language learners. Would the input sentence memory store increase performance or would performance ultimately suffer due to the extra computational load caused by use of the input sentence memory store? The size of the input sentence memory store could also be varied, i.e., an infinite store as in Gold's (1967) paradigm, or bounded by a certain number of sentences. For example, the simulated language learner might only store the last five sentences that it was presented with. This would offset some of the extra computational load that is incurred by checking all of the sentences that the learner has encountered. The input sentence memory store could also be used selectively. For example, only use the memory store when the current input sentence cannot be parsed by a candidate grammar. If the current input sentence cannot be parsed by a candidate grammar there is no reason to take the performance hit incurred by checking the other sentences in

the memory store, since that candidate is not the target language. Adding an input memory store could create a more efficient simulated language learner that is a more accurate model of first language acquisition.

Another area of research would be to perform a comparison study between our SP-compliant learners and Yang's Naïve Parameter Learner (NPL) on the CoLAG domain. The NPL is widely accepted in the research community as a plausible model of first language acquisition. In contrast to our SP-compliant learners, the NPL does not use the Subset Principle to avoid superset hypothesis. It depends on statistics to allow it to "retreat" from superset errors. Pilot studies we have conducted show that the NPL when tested on the CoLAG domain which contains languages that are in subset-superset relationships, takes a substantially larger number of parses to converge on the target grammar than learners that have an SP avoidance strategy 'hardwired' and available for use, though a comprehensive set of simulation runs and analysis of the resulting data remains a project for the future.

In this thesis we only performed a preliminary investigation of the effects of language domain shape on learner performance. There are additional variables that need to be taken into account in order to give a more detailed analysis of the relationship. For example, the subset-superset language domains used in this research do not have properly intersecting languages. Since previous studies have shown that the amount of overlap between languages (a measure of domain ambiguity) can drastically affect learning performance, it seems a reasonable next step to include this as a factor in

future work. In addition, different language domains should be created that vary the amount of intersection between these properly intersecting languages. Varying the connectivity of the lattice is another area to investigate. Each of the rows in the language domains investigated in this thesis were fully connected, i.e., a language in one row was a superset of all of the languages in the row directly below. An interesting target of study would be to see how well the learning efficiency of the SP-compliant learners studied in this thesis fare in domains that exhibited sparsely connected partial orderings of its languages.

Appendix A – Miscellaneous Procedure Descriptions

Environment.GetAnInput():
Selects a sentence at random from the target. Returns the randomly selected sentence.

Environment.SetupTargetLanguage(GrammarID, LanguageDomain)
Gathers all the sentences of the target language and stores them in a member variable of the Environment class.

Flashlight.Reward(GrammarID):
Rewards all supersets of the given grammar.

GetParseGrammarIDs(Sentence):
Gets the set of grammar IDs corresponding to the grammars that can parse the given sentence. Returns a set of grammar IDs.

LanguageDomain.PickRandomGrammar():
Chooses a grammar at random. Returns a grammar ID.

Lattice.GetLargestLanguages():
Gets the set of largest language grammar IDs. Returns a set of grammar IDs.

Lattice.GetSmallestLanguages():
Gets the set of smallest language grammar IDs. Returns a set of grammar IDs.

Lattice.GetSubsetIDs(GrammarID):
Get the subset IDs of a given grammar ID. Returns a set of grammar IDs.

Lattice.IsAMemberOfSL(GrammarID):
Is the given grammar ID a member of the current smallest language set.

Lattice.PickRandomGrammarFromSmallestLanguages():
Randomly picks a grammar from the current set of smallest languages in the lattice. Returns a grammar ID.

Lattice.Remove(GrammarID):
Removes a given grammar from the lattice.

Lattice.RemoveIncludingAllDescendents(Set):
Removes all grammar IDs in the given set from the lattice. All descendents of each grammar ID in the given set are also removed from the lattice.

Learner.PickNextHypothesis(SearchSpace, Environment):
Full descriptions of each learner's override of this procedure will be given later in the dissertation.

Learner.Reset():
Resets a learners state for a new trial. The implementation of this procedure will depend on the learning algorithm. For example, the lattice learners will need to reset the lattice by adding back all removed grammars.

Learner.SetFirstHypothesis():
Sets the initial grammar hypothesis. The implementation of this procedure will depend on the learning algorithm. For example, some of the lattice learners will choose a grammar at random from the smallest language set.

Licensed(Sentence, GrammarID):
Determines whether or not the given grammar can parse the given sentence. Returns true if the grammar can parse the sentence and false otherwise.

Oracle.AttainedTarget(GrammarID):
Determines whether a given grammar ID is the target. Returns true if the given grammar ID is the target or weakly equivalent to the target and false otherwise.

Oracle.Setup(GrammarID):
Gathers grammars that are weakly equivalent to the target grammar.

PickRandomGrammarFromSet(GrammarIDSet):
Picks a grammar at random from a given set of grammar IDs. Returns a grammar ID.

SearchSpace.ResetForTrial():
Resets the search space for a new trial. All grammars in the language domain are put back into the search space.

Set.Add(Element):
Adds the given element to the set.

SetDifference(Set, Set):
Calculates the difference between the two sets. The second set is subtracted from the first. Returns a set.

SetIntersect(Set, Set):
Calculates the intersection of the two given sets. Returns a set.

Simulation::CreateLearner():
Creates an instance of a given learner. The type of learner instance to create is read in from a file.

SPLatticeDecodeFavorUnmarked.PerformSerialParseChoosingUnmarked(Sentence):
Perform a serial parse on the given sentence choosing unmarked values at choice points. Returns the grammar ID of the generated parse.

SPLatticeFlashlight.GetGrammarWithHighestRewardFromSL():
Get the grammar with the highest reward value from the smallest language set. Returns a grammar ID.

TotalOrderingLearner.AddSentence(Sentence):
Adds a sentence to the collection of sentences that have been presented to the learner so far.

TotalOrderingLearner.LicensedAll(GrammarID):
Is the given grammar ID licensed by all the input sentences presented to the learner so far. Returns true if all sentences are licensed by the given grammar ID and false otherwise.

TotalOrdering.MoveToNextGrammarHypothesis():
Moves the current grammar hypothesis pointer to the next item in the enumeration and returns that item. Returns a grammar ID.

Appendix B – Pseudocode Guide

All capitalized words are keywords.

CLASS/ENDCLASS: Used to define a class.

ABSTRACT: Modifies a class definition. Used to define an abstract class. This type of class cannot be instantiated.

INHERITS: Modifies a class definition. Used to define a class that inherits from another class.

VIRTUAL: Modifies a class procedure definition. Used to define a virtual procedure.

CLASSPROCDURES/ENDCLASSPROCEDURES: Block used to define class level procedures.

CLASSVARIABLES/ENDCLASSVARIABLES: Block used to define class level variables.

PROCEDURE/ENDPROCEDURE: Used to define a function or procedure. May or may not return a value.

DECLARE: Declares a variable. Variables declared within a procedure are local to that procedure.

RETURN: Used to return from a procedure.

IF/THEN/ELSE/ENDIF: If statement block. Code is run only if the given boolean condition evaluates to true.

WHILE/ENDWHILE: Looping block. The boolean test is at the beginning.

DO/WHILE: Looping block. The boolean test is at the end. The code is guaranteed to be run at least once.

FOREACH/ENDFOREACH: Looping block. Used to define a for loop.

←: Assignment

/*: Start comment

*/: End comment

References

Angluin, D. (1980). Inductive Inference of Formal Languages from Positive Data. *Information and Control* **45.** 117-135.

Becker, M. (2006). There Began to be a Learnability Puzzle. *Linguistic Inquiry* **37.** 441-456.

Bertolo, S. (2001). A Brief Overview of Learnability. In S. Bertolo (Ed.), *Language Acquisition and Learnability.* 1-14. Cambridge, UK: Cambridge University Press.

Berwick, R. (1985). *The Acquisition of Syntactic Knowledge.* Cambridge, MA: MIT Press.

Berwick, R. & Niyogi, P. (1996). Learning From Triggers. *Linguistic Inquiry* **27.** 605-622.

Briscoe, E. (2000). Grammatical Acquisition: Inductive Bias and Coevolution of Language and the Language Acquisition Device. *Language* **76.** 245-296.

Chomsky, N. (1957). *Syntactic Structures.* Mouton: The Hague.

Chomsky, N. (1980). On Binding. *Linguistic Inquiry* **11.** 1-46.

Chomsky, N. (1981). *Lectures on Government and Binding.* Dordrecht: Foris.

Chomsky, N. (1982). *Some Concepts and Consequences of the Theory of Government and Binding.* Cambridge, MA: MIT Press.

Chomsky, N. (1986). *Knowledge of Language: Its Nature, Origin, and Use.* New York: Praeger.

Clark, R. (1992). The Selection of Syntactic Knowledge. *Linguistic Acquisition* **2.** 83-149.

Fodor, J. (1998a). Parsing To Learn. *Journal of Psycholinguistic Research* **27.** 339-374.

Fodor, J. (1998b). Unambiguous Triggers. *Linguistic Inquiry* **29.** 1-36.

Fodor, J., Melnikova, Y. & Troseth, E. (2002). A Structurally Defined Language Domain for Testing Syntax Acquisition Models. *CUNY-CoLAG Working Paper #1*. City University of New York. At the website of the CUNY Computational Language Acquisition Group

Fodor, J. & Sakas, W. (2004). Evaluating Models of Parameter Setting. In A. Brugos, L. Miciulla & C. Smith (Eds.), *BUCLD 28: Proceedings of the 28th Annual Boston University Conference on Language Development*, Cascadilla Press, Somerville, MA, 1-27.

Fodor, J. & Sakas, W. (2005). The Subset Principle in Syntax: Costs of Compliance. *Journal of Linguistics* **41.** 1-57.

Fodor, J., Sakas, W. & Hoskey, A. (2007). Implementing the Subset Principle in Syntax Acquisition: Lattice-Based Models. In S. Vosniadou, D. Kayser & A. Protopapas (Eds.), *Proceedings of the European Cognitive Science Conference 2007*, Taylor and Francis, Delphi, Greece.

Fodor, J. & Teller, V. (2000). Decoding Syntactic Parameters: The Superparser as Oracle. *Proceedings of the Twenty-Second Annual Conference of the Cognitive Science Society*, Philadelphia, PA, 136-141.

Gibson, E. & Wexler, K. (1994). Triggers. *Linguistic Inquiry* **25.** 407-454.

Gold, M. (1967). Language Identification in the Limit. *Information and Control* **10.** 447-474.

Inoue, A. & Fodor, J. (1995). Information-Paced Parsing of Japanese. In Mazuka & Nagai (Eds.), *Japanese Sentence Processing.* Hillsdale, NJ: Lawrence Erlbaum.

Joshi, A. (1994). Commentary: Some Remarks on the Subset Principle. In B. Lust, G. Hermon & J. Kornfilt (Eds.), *Syntactic theory and first language acquisition: cross-linguistic perspectives (vol. 2).* Hillsdale, NJ: Larwrence Erlbaum.

Kapur, S., Lust, B., Harbert, W. & Martohardjono, G. (1993). Universal Grammar and Learnability Theory: The Case of Binding Domains and the 'Subset Principle'. In R. Reuland & W. Abraham (Eds.), *Knowledge and Language, Volume I, From Orwell's Problem to Plato's Problem.* Dordrecht: Kluwer.

MacLaughlin, D. (1995). Language Acquisition and the Subset Principle. *Linguistic Acquisition* **12.** 143-191.

Manzini, R. & Wexler, K. (1987). Parameters, Binding Theory, and Learnability. *Linguistic Inquiry* **18.** 413-444.

Niyogi, P. & Berwick, R. (1996). A Language Learning Model for Finite Parameter Spaces. *Cognition* **61.** 161-193.

Pinker, S. (1979). Formal Models of Language Learning. *Cognition* **7.** 217-283.

Sakas, W. (2000a). *Ambiguity and the Computational Feasibility of Syntax Acquisition.* Doctoral Dissertation,City University of New York.

Sakas, W. (2000b). Modeling the Effect of Cross-Language Ambiguity on Human Syntax Acquisition. In *Proceedings of the joint meeting of the 4th Computational Natural Language Learning Workshop (CoNLL-2000) 2nd Learning Language in Logic Workshop and 5th International Colloquium on Grammatical Inference*, Association for Computational Linguistics, Lisbon.

Sakas, W. (2003). A Word-Order Database for Testing Computational Models of Language Acquisition. In E. Hinrichs & DanRoth (Eds.), *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, Sapporo, Japan, 415-422.

Sakas, W. & Fodor, J. (1998). Setting the First Few Syntactic Parameters - A Computational Analysis. In *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society*, Lawrence Erlbaum Associates.

Sakas, W. & Fodor, J. (2000). Setting Syntactic Parameters: A Computational Analysis of Child-directed Speech. *CUNY Collaborative Incentive Research Grant*. City University of New York.

Sakas, W. & Fodor, J. (2001). The Structural Triggers Learner. In S. Bertolo (Ed.), *Language Acquisition and Learnability.* 172-233. Cambridge, UK: Cambridge University Press.

Sakas, W. & Nishimoto, E. (2002). Search, Structure or Statistics? A Comparative Study of Memoryless Heuristics for Syntax Acquisition. In *Proceedings of the 24th Annual Conference of the Cognitive Science Society*, Fairfax, VA, 786-791.

Valian, V. (1990). Logical and psychological constraints on the acquisition of syntax. In L. Frazier & J. Villiers (Eds.), *Language Processing and Language Acquisition.* Dordrecht: Kluwer.

Wexler, K. (1993). The Subset Principle is an Intensional Principle. In E. Reuland & W. Abraham (Eds.), *Knowledge and Language: Issues in Representation and Acquisition.* 217-239. Dordrecht: Kluwer.

Wexler, K. & Culicover, P. (1980). *Formal Principles of Language Acquisition.* Cambridge, MA: MIT Press.

Wexler, K. & Manzini, R. (1987). Parameters and Learnability in Binding Theory. In T. Roeper & E. Williams (Eds.), *Parameter Setting.* 41-76. Dordrecht: Reidel.

Yang, C. (2002). *Knowledge and learning in natural language.* New York: Oxford University Press.