Vectorial Representations of Meaning for a Computational Model of Language Comprehension

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Stephen Tze-Inn Wu

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor Of Philosophy

June, 2010

Acknowledgements

First, I want to thank my family: my wife, Fran, for her love, companionship, and daily prayer support throughout the best and hardest times. My mother and father, Tim and Tai, have nurtured and encouraged and prayed for me relentlessly. I thank my brother, Sam, for being a sounding board and just the kind of sibling that a researcher needs. And I thank my late grandmother for teaching me character and persistence alongside childlike faith. Without my family, I would not be the person who wrote this thesis.

Many people have contributed to my growth as a researcher. I would like to thank Professor Schuler for hours of discussion and collaboration. His pursuit of precision and excellence in research, and his ability to enjoy himself in the meanwhile, has set an incredible example for me. I have also learned from the other members of the NLP lab at the University of Minnesota: Tim Miller, Lane Schwartz, Andy Exley, Dingcheng Li, and Luan Nguyen. Their help and collaboration on the big projects to the smallest questions have sped my growth immensely. I want to thank my thesis committee, as well, for being a wise and encouraging community: Professor Boley, Professor Fletcher, Professor Gini, and Dr. Savova.

Finally, I would like to thank Jesus Christ, my savior and treasure. There is no other source of sustaining strength that I could have relied upon, no greater comfort in the difficult times; and there is no deeper joy than following him, even through a Ph.D. program.

Dedication

To the one who is worthy

ABSTRACT

This thesis aims to define and extend a line of computational models for text comprehension that are humanly plausible. Since natural language is human by nature, computational models of human language will always be just that — models. To the degree that they miss out on information that humans would tap into, they may be improved by considering the human process of language processing in a linguistic, psychological, and cognitive light.

Approaches to constructing vectorial semantic spaces often begin with the distributional hypothesis, i.e., that words can be judged 'by the company they keep.' Typically, words that occur in the same documents are similar, and will have similar vectorial meaning representations. However, this does not in itself provide a way for two distinct meanings to be composed, and it ignores syntactic context.

Both of these problems are solved in *Structured Vectorial Semantics* (SVS), a new framework that fully unifies vectorial semantics with syntactic parsing. Most approaches that try to combine syntactic and semantic information will either lack a cohesive semantic component or a full-fledged parser, but SVS integrates both. Thus, in the SVS framework, interpretation is interactive, considering both syntax and semantics simultaneously.

Cognitively-plausible language models would also be incremental, support linear-time inference, and operate in only a bounded store of short-term memory. Each of these characteristics is supported by right-corner Hierarchical Hidden Markov Model (HHMM) parsing; therefore, SVS will be transformed into right-corner form and mapped to an HHMM parser. The resulting representation will then encode a psycholinguistically plausible incremental SVS language model.

Contents

Acknowledgements **i**

Dedication **ii**

Abstract **iii**

List of Tables **viii**

List of Figures **ix**

1 Introduction **1**

1.1 Goals of this Thesis Work 1

1.2 Contributions of this Thesis Work 3

1.3 Chapter Descriptions 3

I Language Comprehension with Structured Vectorial Semantics **6**

2 Theoretical Foundations for Structured Vectorial Semantics **7**

2.1 Linear algebra 8

2.2 Grammars and Trees 9

2.3 Syntactic Parsing 12

2.4 The CKY Parser 14

2.5 Headword Lexicalization 16

3 Structured Vectorial Semantics **18**
3.1 Syntactic and Semantic Dependencies 19
3.2 Vectors, Probabilities, and Denotations 21
3.3 Structured Vectorial Semantic Parsing 24
3.4 Bilexical Parsing with SVS 28
3.5 Empirical Validation of SVS Assumptions 30
3.5.1 Effect of SVS Grammar Factorization 31
3.5.2 Effect of Semantic Vectorization on Accuracy 32
3.6 Summary 33

4 Relational Clustering with SVS **34**
4.1 Language Models in SVS Instantiations 36
4.2 Inducing Relational Clusters 37
4.3 Relational Semantic Clusters in Parsing 39
4.4 Evaluation 40
4.4.1 Implementation 40
4.4.2 Interpretable and cohesive relational clusters 42
4.4.3 Dependency on Initial Values 42
4.4.4 Comparison to Lexicalization 43
4.4.5 Effect of Vectorization on Speed 44
4.4.6 Effect of EM Iterations 45
4.4.7 Effect of Number of clusters 45
4.5 Summary and Discussion 46

5 Logical Interpretation with SVS **48**
5.1 Vectors and Matrices as Sets 49
5.2 Logical operators 51
5.3 SVS Language Model for Logical Interpretation 54
5.4 A Domain-specific Example 54
5.5 Summary 59

II Plausible Language Models with SVS **61**

6 Right-Corner Syntactic Parsing **62**
6.1 Right-corner transform 62
6.2 Hidden Markov Models 67
6.3 Hierarchic Hidden Markov Models 68
6.4 Right-corner trees and HHMMs 71
6.5 Summary 76

7 Incremental Structured Vectorial Semantics **77**
7.1 Incremental Syntactic Parsing with HHMMs 78
7.2 Mapping Structured Vectorial Semantics to HHMM Parsing 84
7.3 Discussion 89
7.3.1 Predictions 90
7.3.2 Corpus Analysis 91
7.4 Summary 92

8 Language Modeling Principles **94**
8.1 The Existence of Interactive Semantics 95
8.2 Mental Representations of Meaning 97
8.3 Manner of Interpretation 103
8.4 Summary 107

9 Related Work **109**
9.1 Semantic Spaces 109
9.2 Syntactic Processing 112
9.3 Combining Syntax and Semantics 114
9.4 Interpretation of Natural Language 117
9.5 Summary 119

10 Conclusion **120**

References **122**

Appendix A. Proofs and Derivations **134**

A.1 Derivation of incremental SVS HHMM Probabilities 134

List of Tables

2.1 A summary of notation from Chapter 2. 9
3.1 Structured vectorial semantics notation. 22
3.2 Effect of vectorization on parsing accuracy 31
4.1 Example clusters from θ_{H} 41
4.2 Effect of EM random initialization on relationally-clustered SVS 42
4.3 Parsing accuracy in relationally-clustered SVS 43
4.4 Effect of EM iterations on parsing accuracy 45
7.1 Comparison of right-corner parsing equations 84
7.2 Corpus distribution of determiners over memory element use 91

List of Figures

2.1 Sample grammar, tree, and bracketed sentence 11
2.2 CKY Algorithm Pseudocode 15
2.3 Sample parse chart from a CKY parser 16
2.4 Sample tree with headwords h 17
3.1 Sample tree with semantic referents e and relations l 19
4.1 Probability models needed as input for SVS 36
4.2 Speed of relationally-clustered SVS with and without vectorization 44
4.3 Effect of number of clusters on parsing accuracy 46
5.1 Graphical and vectorial representations of a sample world model 55
5.2 Sample tree and grammar for logical-interpretation SVS syntax 56
5.3 Logical-interpretation SVS as an operator tree 58
5.4 Logical-interpretation SVS as a linear operator chain 59
6.1 Right-corner transformed tree with pre-transform addresses 64
6.2 Right-corner transform formal rewrite rules 66
6.3 Graphical representation of a Hidden Markov Model. 67
6.4 Graphical representation of a Hierarchical Hidden Markov Model 69
6.5 Sample right-corner transformed tree 72
6.6 HHMM parsing as time-unfolded stack operations 73

Chapter 1

Introduction

The human capacity for producing and understanding language involves a dizzying array of factors. From the neural level of brain activation to the perceptual level of our senses; from the symbolic level of grammatical analysis to the subjective nature of meaning in context — the possibility for miscommunication seems immense. Yet we speak and we write, and people understand.

In each of us, linguistic analysis must interface with the physical world at one end (in speech or text) and communicate something meaningful to our minds at the other end. Knowingly or not, we incorporate linguistic phenomena into our communication that allow our listeners (and readers) to understand us better. And when we seek to understand others, we don't just hear *words*; we construct a coherent *mental image* of the speaker's or author's original intent.

Considering that this process in humans can be so highly complex yet efficient, this thesis advocates for an approach to computationally understanding language that draws on how humans 'do language.'

1.1 Goals of this Thesis Work

The long-term goal in this direction of research is to develop a computational model of language comprehension — a model that integrates all levels of linguistic analysis, reflects human processing of language, and can be empirically viable in texts of any natural language. That is to say: it should do what humans do.

Studies in linguistics, cognitive science, and psychology direct us to human-like language models that *incrementally* form and refine meaning in *natural language texts* using *vectorial representations* in an *integrated*, *probabilistic* model of language.

Natural language texts. In contrast to a limited, artificial languages that are recognized by many user interfaces, the language models in this thesis will consider their input to be natural language (i.e., one spoken and written by humans for natural communication). The focus in evaluations is text, and the output is a 'mental image' — a meaning representation of the text as it is recognized in the reader's mind. Strategies applied to text may also be relevant for speech.

Incremental account of meaning. An ideal meaning representation is built up and modified as recognition proceeds from left to right, just as humans build mental representations of a text as they read from left to right.[1] Stopping in the middle of a sentence, a computational model's meaning representation should mimic a person's clarity or ambiguity as to what is being talked about. As new words are seen, their meaning should cohere with the existing representation.

Vectorial representations of meaning. Mental representations are in the form of vectors and matrices, and may be composed from other units of meaning. Linguistically, this is like building a mental representation out of some pre-defined, basic building blocks of meaning; psychologically, each mental representation might correspond to some pattern of activity in neurons. 'Gist' meanings of this kind are then defined and composed using the tools of linear algebra. There is good evidence that we use a similar mechanism in our minds (Johnson-Laird, 1987; Landauer and Dumais, 1997; Smolensky and Legendre, 2006).

Probabilistic model. Probabilistic frameworks have been shown to be effective in modeling accurate recognition of linguistically ambiguity utterances. Both parsing algorithms presented in this work (CKY and HHMM) allow for competing hypotheses to be represented and for the most likely hypothesis to be chosen. In addition, vectorial representations of meaning will be embedded inside standard probabilistic phrase structure rules. Therefore, the resulting semantic representations will be compatible with statistical parsers.

[1] Of course, many languages are read right to left or up to down. But for the purposes of this paper, both texts and time will exclusively be referred to as proceeding from left to right.

Integrated language model. A parser must have a basis on which to make its decisions about likely syntactic structure (and in our case, semantic meaning). A probability distribution that describes how such structure can be generated is a *language model*. In this thesis, the language model can include factors from grammar (morphology and syntax are smashed together in this model) and semantics (the vector-based meanings discussed above). The model could be extended to account for speech, so that all parts of language production and recognition are modeled.

1.2 Contributions of this Thesis Work

The main contributions of this thesis are:

- Structured Vectorial Semantics (SVS), a novel language comprehension framework in which syntax and semantics are fully interactive. Other models have semantic vectors interact with syntactic context; however, in SVS, vector composition necessarily produces hypothetical parses. Broad-coverage parsing that accomplishes compositional semantics is therefore possible with SVS.

- Relational-clustering SVS, an instantiation of SVS that uses word clusters in head and modifier relationships. The main empirical results of the thesis are that the vectorial semantics of relational-clustering SVS improve syntactic parsing accuracy and speed.

- Incremental SVS, an extension of SVS to allow for incremental interpretation and a model of bounded short-term memory during sentence processing. Unlike other techniques that bear similar psycholinguistically-plausible characteristics, incremental SVS is amenable to broad-coverage NLP. As a comprehensive model of language comprehension, it makes interesting predictions about quantifier distribution in English.

1.3 Chapter Descriptions

This thesis is organized into two parts. Part I introduces a language model called *Structured Vectorial Semantics*. This new framework may be considered a syntactic

extension to vector-space models, or a vector-semantic extension to standard statistical parsing techniques.

- **Chapter 2**. Since formalisms from many fields are brought together in Structured Vectorial Semantics, this is a reference chapter of introductions to linear algebra, formal languages, and parsing. Only concepts and operations necessary for SVS are presented. For some readers, this chapters may be skipped or skimmed — their notation is summarized in Table 2.1.

- **Chapter 3**. The Structured Vectorial Semantic framework is described in full here. First, semantic 'concepts' and 'relations' are introduced, which are considered alongside syntax during parsing. The integrated syntactic–semantic model is then vectorized. This chapter is the main theoretical contribution of this thesis work, representing a language modeling assumption that syntax and semantics are distinct yet simultaneous.

- **Chapter 4**. Despite the fact that SVS represents semantics within vectors, what goes into the vectors (meaning representation) needs to be instantiated. Adapting from the word co-occurrence and dimensionality reduction ideas of common vector-space models, relationally-clustered versions of headwords are used here as vector elements. The main empirical results of this thesis are in this chapter, showing that substantial parsing accuracy gains can be obtained by using relationally-clustered headwords in SVS.

- **Chapter 5**. Sometimes, more specific semantic knowledge is known (or can be learned) than the 'gist'-style meaning of vector-space dimensionality-reduction models. For precise information in the form of first-order predicate logic, this chapter shows by way of definition and example that SVS is able to handle semantic interpretation.

Part II describes how Structured Vectorial Semantics fits in with psycholinguistically plausible language modeling assumptions.

- **Chapter 6**. As a background reference to Chapter 7, this chapter covers the right-corner transform as it operates on grammar trees. It also describes Hierarchic Hidden Markov Models and how they may be used to parse right-corner trees.

- **Chapter 7**. Syntactic CFGs may be transformed into an incremental form and then parsed; the same transformation may be done for SVS, which was defined on PCFGs. This yields a novel parser which can do incremental interpretation. Defining logical interpretation as in Chapter 5 in this incremental framework leads to some interesting predictions on how quantifiers interact with short-term memory. These linguistic predictions are validated through a corpus study.

- **Chapter 8**. Psycholinguistically-motivated modeling assumptions and principles guided the development of Structured Vectorial Semantics and its transformation into right-corner form. The resulting language models allow interaction between semantics and other linguistic processes and seek to interpret and store meaning in a manner similar to models in cognitive science.

- **Chapter 9**. Because Structured Vectorial Semantics is an interdisciplinary framework, inspirational approaches and similar work in several fields are discussed. It is shown that SVS extends traditional vector-space models with syntax, and traditional parsing models with semantics. Also, a review of other joint approaches to syntax and semantics and systems concerned with language understanding is given.

- **Chapter 10**. This final chapter summarizes the contributions of the thesis.

It is my hope that this thesis work will spur on other psychologically-plausible, empirically-useful models of language and cognition, and that in seeking to define such models we may gain some insight into the incredible mystery that language is.

Part I

Language Comprehension with Structured Vectorial Semantics

Chapter 2

Theoretical Foundations for Structured Vectorial Semantics

The models of language in this thesis require theory from a number of different fields. The sections in this chapter are intended to give relevant information from different disciplines so that the proposed language models can be understood. Thus, sections will begin with some basics for readers outside each field, then build toward the specific constructs used later in this thesis.

For an introduction to linear algebra, Section 2.1 gives some basic definitions, operations, and notation for vectors and matrices; some notation particular to this thesis is also introduced. NLP researchers may skip Section 2.2, which introduces the concepts of grammars and trees. Section 2.3 builds on these concepts with for a definition of syntactic parsing with a notation that will be useful later in Chapter 7. Next, Section 2.4 details the standard CKY algorithm, which is used for the non-incremental SVS implementations of Part I. Finally, Section 2.5 defines lexicalization as is often used in parsing techniques.

2.1 Linear algebra

A *vector* is an ordered set of numbers or variables, which can be a row or vector:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}, \qquad \mathbf{v} = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$$

Matrices have elements with two distinct indices. One way to view them is as a concatenation of vectors:

$$\mathbf{U} = \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,m} \\ u_{2,1} & u_{2,2} & \cdots & u_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n,1} & u_{n,2} & \cdots & u_{n,m} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m \end{bmatrix}$$

Conversely, a vector may be viewed as a degenerate matrix.

Vectors and matrices support several operations, which will be shown on matrices here; vectorial counterparts to these definitions can be derived by considering vectors to be $n \times 1$ matrices. The *transpose* of a matrix $\mathbf{U}$, denoted $\mathbf{U}^T$, flips the matrix so that rows become columns and columns become rows (compare the elements to those of $\mathbf{U}$):

$$\mathbf{U}^T = \begin{bmatrix} u_{1,1} & u_{2,1} & \cdots & u_{n,1} \\ u_{1,2} & u_{2,2} & \cdots & u_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ u_{1,m} & u_{2,m} & \cdots & u_{n,m} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \vdots \\ \mathbf{u}_m^T \end{bmatrix}$$

The *inner product* (or dot product) is the most common type of product, and can be done on matrices or vectors. Sums are over both j and k (inner) indices:

$$\mathbf{UV} = \begin{bmatrix} \sum u_{1,j}\, v_{k,1} & \sum u_{2,j}\, v_{k,1} & \cdots & \sum u_{n,j}\, v_{k,1} \\ \sum u_{1,j}\, v_{k,2} & \sum u_{2,j}\, v_{k,2} & \cdots & \sum u_{n,j}\, v_{k,2} \\ \vdots & \vdots & \ddots & \vdots \\ \sum u_{1,j}\, v_{k,m} & \sum u_{2,j}\, v_{k,m} & \cdots & \sum u_{n,j}\, v_{k,m} \end{bmatrix}$$

The *outer product* (or cross product) takes two vectors and produces a matrix. To show why it is termed such, elements of each vector are written as matrices; the outer dimensions are retained in the resulting matrix. Here, $\mathbf{u}$ is a column vector, $\mathbf{w}$ is a row vector:

$$\mathbf{u}_{[n\times 1]} \times \mathbf{w}_{[1\times m]} = \begin{bmatrix} u_{1,1}\, w_{1,1} & u_{1,1}\, w_{1,2} & \cdots & u_{1,1}\, w_{1,m} \\ u_{2,1}\, w_{1,1} & u_{2,1}\, w_{1,2} & \cdots & u_{2,1}\, w_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n,1}\, w_{1,1} & u_{n,1}\, w_{1,2} & \cdots & u_{n,1}\, w_{1,m} \end{bmatrix}$$

Symbol	Description
Probability notation	
$\mathsf{P}(\cdot)$	Probability of the argument, between 0 and 1; or, a distribution defining such probabilities
$\mathsf{P}(\cdot,\cdot)$	Joint probability that both arguments occur; or, a joint distribution
$\mathsf{P}(\cdot\,\vert\,\cdot)$	Conditional probability, where '|' can be read "given that"; or, a conditional distribution
$\tilde{\mathsf{F}}(\cdot)$	A frequency count of the given joint state
Θ	Probability distribution from which a parenthesized probability is drawn
Linear Algebra (Section 2.1)	
$\mathbf{u}$	Vector or tensor (in matrix form) indicated by bold; e.g., $\mathbf{e}$
$\mathbf{U}$	Matrices indicated by uppercase sans-serif font; e.g., $\mathbf{L}$
$\mathbf{u}^T, \mathbf{U}^T$	Transpose. Columns become rows, rows become columns
$d(\mathbf{u})$	Diagonal matrix made from the vector $\mathbf{u}$
$\mathbf{UV}$	Matrix multiplication (inner product); rows of $\mathbf{U}$ are multiplied by columns of $\mathbf{V}$ and summed
Syntactic notation (Section 2.2–2.4)	
x	Yield, i.e., the observed string; at a terminal in a tree, this is a word
g	Non-terminal node in a grammar tree, representing the left-hand side of a grammar rule
c	Syntactic constituent or POS tag
τ	Grammar tree or subtree
$\mathscr{D}$	Dictionary; the unary section of a CNF grammar (terminals on the right)

Table 2.1: A summary of notation from Chapter 2.

A *pointwise product* can be accomplished between two column length-n vectors $\mathbf{u}$ and $\mathbf{v}$ using the above operations. Matrices that contain non-zero elements only on the diagonal (from top left to bottom right) are called *diagonal matrices*. Listing the elements of $\mathbf{u}$ on the diagonal of a matrix (denoted $d(\mathbf{u})$), the pointwise product is obtained via the inner product.

$$d(\mathbf{u}) \cdot \mathbf{v} = \begin{bmatrix} u_1 & 0 & \cdots & 0 \\ 0 & u_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & u_n \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_1\, v_1 & u_2\, v_2 & \cdots & u_n\, v_n \end{bmatrix}$$

Though the content of the vectors differs among the approaches in this dissertation, these linear algebra definitions and operations are sufficient to fully define the vectorial theory of meaning.

2.2 Grammars and Trees

In much of linguistics, texts are considered a sentence at a time. The fields of Morphology (how words are formed and modified in sentences) and Syntax (how words are

ordered in sentences) describe most of the sentence-level phenomena that are observable in text. Natural Language Processing techniques often ignore morphology or make assumptions implicitly, and only deal with syntax. For instance, *Lucy's_cat* might be expanded to *Lucy_'s_cat*. This assumes that the possession marker (*'s*) can be split off from the word itself and be reordered just like other words (which happens to be close to the truth, since it could be considered a phrasal affix). So the discussion below only considers how 'words' are ordered in a sentence.

A sentence with n words is essentially a sequence of the words, $w_1, w_2, \ldots, w_n$. In corpora of linguistically-annotated sentences, each word is usually labeled with its part of speech (POS). In this dissertation, these labels will resemble the Wall Street Journal (WSJ) corpus POS tags. For example,

The	*boy*	*who*	*cried*	*wolf*	*looked*	*around*	.
DT	NN	WHNP	VBD	NN	VBD	RB	.

Determiners (DT), nouns (NN), relative pronouns (WHNP), participial verbs (VBN), past-tense verbs (VBD), and adverbs (RB) are just some of the many parts of speech that can describe the function of a word in a sentence.

But there are larger units of sentence structure than just parts of speech. *The boy who cried wolf* is a whole phrase that refers to one person. So, noun phrases (NP), verb phrases (VP), adjective phrases (ADJP), and the like combine words into phrases; ventually, phrases can similarly be combined into sentences. Things like parts of speech and phrases are called syntactic *categories* or *constituents*.

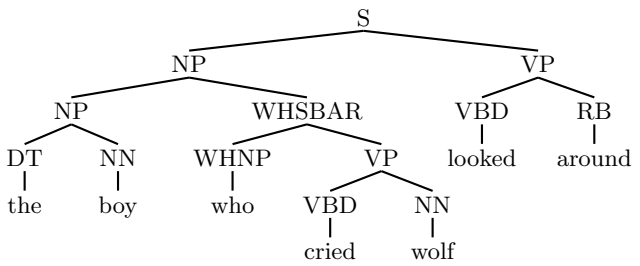
For a given language, *phrase structure rules* can be written that determine which phrases and parts of speech are permissible in what order. The phrase structure rules for a language, when taken together, constitute a *formal grammar*, or just *grammar*. This provides a way to generate a valid sentence in the langauge; or, taken another way, it defines exactly what sequences of words are permissible in the language.

A *Context-free Grammar* (CFG) is a useful type of grammar that has rules in which the constituent on the left of the rule has only one symbol and therefore does not depend on its context (e.g., the noun can be replaced by "boy" or "wolf" no matter what the determiner is). This is contrasted with a Context-sensitive Grammar, which would have multiple symbols on the left, and would therefore depend on its context.

a)

S → NP VP
NP → NP WHSBAR
NP → DT NN
WHSBAR → WHNP VP
VP → VBD NN
VP → VBD RB
DT → the
NN → boy
NN → wolf
WHNP → who
VBD → looked
VBD → cried
RB → around

b)



c)

[S [NP [NP [DT the] [NN boy]] [WHSBAR [WHNP who] [VP [VBD cried] [NN wolf]]] [VP [VBD looked] [RB around]]]]

Figure 2.1: a) A toy English grammar. b) A possible tree for the given sentence, consistent with the grammar. c) A bracketed sentence with the same information as b).

In *Chomsky Normal Form* (CNF), a CFG rule can have only binary-branching rules — rules with exactly two constituents on the right side. The only exception to the two-on-the-right restriction is that words are generated by a unary rule from their POS. So acceptable rules are of the form

$$g_1 \rightarrow g_2 \quad g_3 \tag{2.1}$$

$$g_1 \rightarrow x \tag{2.2}$$

where x is observed input (words) and g are non-terminals (anything else).

Any CFG can be rewritten in Chomsky Normal Form via several binarization strategies. In this work, binarization is done in a manner which preserves lexical heads (Magerman, 1995). An example CNF grammar is shown in Figure 2.1a. The last part of the grammar defines the lexicon, which is essentially a dictionary that is defined in terms of CNF's unary rules.

Figure 2.1b shows a tree representation of particular sentence. Trees, denoted by T, naturally show the hierarchical structure inherent in grammars. A category on the left side of a rule is called a *parent*, a category on the right is called a *child*. Each tree shows a single possible way of grouping the words such that they form phrases and then sentences, though more than one may be possible in the grammar.

The bracketed sentence in Figure 2.1c is equivalent to the tree in Figure 2.1b. This convention of bracketing is the format for corpora of tree-annotated sentences such as

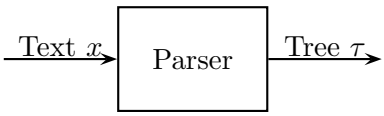
the WSJ corpus, as well as the output of most parsers.

Now, the goal of parsing can be defined in terms of the words and trees (assumed to be acceptable in the grammar). In this light, parsing is the task of choosing between different possible trees that are permissible in the grammar.

Because of the ambiguity of syntax, it is common to define an extra source of information: the probability that a particular grammar rule expands a constituent. With such probabilities annotated on each rule in a CFG, it is now a *probabilistic* context-free grammar (PCFG). Probabilities are typically estimated from some large corpus of sentences annotated with the correct tree; the WSJ corpus was created for this specific purpose, and will be used for syntactic rules in this thesis. An empirical count is made of the possible children of each parent in all the trees of the corpus; dividing this count by the total number of children for that parent gives a conditional probability, $\mathsf{P}(g_1, g_2, g_3) \,/\, \mathsf{P}(g_1) = \mathsf{P}(g_1 \rightarrow g_2\, g_3)$.

2.3 Syntactic Parsing

The task of parsing is to discover grammatical structure, usually in the form of trees τ, from observed input text x.



The constructing of the tree τ is done from grammar rules. This thesis will denote syntactic categories as c and string yields (i.e., observed input) as x. The location of these variables in phrase structure will be identified using subscripts that describe the path from the root to the constituent.[1] Paths consist of left branches (indicated by '0's in the path) and right branches (indicated by '1's), concatenated into sequences η (or ι), and ϵ is the empty sequence at the root. So for a syntactic constituent c_η with path η, that constituent's left child would be identified by $c_{\eta 0}$, and the right child would be identified by $c_{\eta 1}$. PCFG rules are then each of the form $c_\eta \rightarrow c_{\eta 0}\, c_{\eta 1}$ with some probability.

[1] For simplicity, trees are assumed to be compiled into strictly binary-branching form.

A parsing model that directly followed the diagram above would be a *discriminative* approach, where finding the best tree $\hat{\tau}$ is based on a direct estimation of tree probabilities given the input $\mathsf{P}(\tau_\epsilon \,|\, x)$. *Generative* approaches to parsing instead make use of Bayes' rule and give the tree that can best generate the input:

$$\operatorname*{argmax}_{\tau_\epsilon} \mathsf{P}(\tau_\epsilon \,|\, x) = \operatorname*{argmax}_{\tau_\epsilon} \frac{\mathsf{P}(x \,|\, \tau_\epsilon)\mathsf{P}(\tau_\epsilon)}{\sum_x \mathsf{P}(x \,|\, \tau_\epsilon)\mathsf{P}(\tau_\epsilon)} = \operatorname*{argmax}_{\tau_\epsilon} \mathsf{P}(x \,|\, \tau_\epsilon)\mathsf{P}(\tau_\epsilon) \tag{2.3}$$

where the denominator in the second equality is inconsequential because the argmax operation is independent of τ_ϵ.

Using PCFGs in the generative framework, the input x_η is the *yield*, i.e., the observed (sub)string which eventually results from the progeny of constituent c_η; multiple trees τ_η can then be constructed at η by stringing together different grammar rules that are consistent with observed text.

Beginning from preterminal nodes, an initial estimate is made of how likely a syntactic category would produce the observed word, $\mathsf{P}_{\theta_{\text{P-Vit(G)}}}(x_\eta \,|\, c_\eta)$. For a grammar in CNF, trees join preterminals and other nonterminals via binary branches. Thus, for a hypothetical tree rooted at η with category c_η and yield x_η, we will find substring probabilities and trees by using probabilities according to the PCFG model θ_{G}.

$$\mathsf{P}_{\theta_{\text{G}}}(c_\eta \to c_{\eta 0}\ c_{\eta 1}) = \mathsf{P}_{\theta_{\text{G}}}(c_{\eta 0}\ c_{\eta 1} \,|\, c_\eta) \tag{2.4}$$

Notice that although we can view this as 'joining' nonterminals together, the probabilities actually describe the likelihood of generating the nonterminal children from their parent.

Any yield x_η can be decomposed into prefix $x_{\eta 0}$ and suffix $x_{\eta 1}$ yields:

$$x_\eta = x_{\eta 0}\, x_{\eta 1} \tag{2.5}$$

or conversely, the prefix and suffix can be concatenated to give the yield at a parent nonterminal. To find the probability of a best subtree, i.e., the *Viterbi* probability, the highest probability of all yield decompositions is chosen:

$$\mathsf{P}_{\theta_{\text{Vit(G)}}}(x_\eta \,|\, c_\eta) = \max_{x_{\eta 0} c_{\eta 0}, x_{\eta 1} c_{\eta 1}} \mathsf{P}_{\theta_{\text{G}}}(c_\eta \to c_{\eta 0}\ c_{\eta 1}) \cdot \mathsf{P}_{\theta_{\text{Vit(G)}}}(x_{\eta 0} \,|\, c_{\eta 0}) \cdot \mathsf{P}_{\theta_{\text{Vit(G)}}}(x_{\eta 1} \,|\, c_{\eta 1}) \tag{2.6}$$

and the corresponding tree $\hat{\tau}_\eta$ can be constructed from best child trees $\hat{\tau}_{\eta 0}$ and $\hat{\tau}_{\eta 1}$. This recursive definition takes into account binary θ_{G} grammar rule applications, and ends

at the preterminal probability model $\theta_{\text{P-Vit(G)}}$. Note that the maximization over $x_{\eta 0}$ and $x_{\eta 1}$ does not scope over all possible yields, but rather the possible observed yield spans subsumed by $c_{\eta 0}$ and $c_{\eta 1}$.

Given some prior model $\mathsf{P}_{\pi_{\text{G}\epsilon}}(c_\epsilon)$, the probability at the root node can be obtained:

$$\mathsf{P}(x\, c_\epsilon) = \mathsf{P}_{\theta_{\text{Vit(G)}}}(x \mid c_\epsilon) \cdot \mathsf{P}_{\pi_{\text{G}\epsilon}}(c_\epsilon) \tag{2.7}$$

The Viterbi probability $\mathsf{P}_{\theta_{\text{Vit(G)}}}(x \mid c_\epsilon)$ that maximizes this root probability will have an associated tree $\hat{\tau}_\epsilon$ that includes the root syntactic category c_ϵ; this tree can be constructed by referring back to best subtrees at its children, $\hat{\tau}_0$ and $\hat{\tau}_1$.

2.4 The CKY Parser

A standard method for parsing sentences with PCFGs is the CKY (or CYK) algorithm, named for its inventors Cocke, Younger, and Kasami. It is a dynamic programming algorithm that searches possible PCFG rule expansions to find all trees τ_ϵ explaining the observed input x_ϵ. With the probabilities in the PCFG, the best tree $\hat{\tau}_\epsilon$ is found from the Viterbi probabilities just as in Equation 2.6.

Figure 2.2 gives pseudocode for the algorithm. The parser works bottom up, first finding valid preterminals for each word according the probability model $\theta_{\text{P-Vit(G)}}$.

Next, it uses a dynamic programming algorithm to store the best explanations of every substring (from j to $j+i$) of data. This involves picking the best place to split the string into left and right child (i.e., choosing k in the algorithm, or yield spans $x_{\eta 0}$ and $x_{\eta 1}$ in the equations) and the best grammatical constituent for the substring (i.e., choosing c_η). For this best grammatical constituent, associated backpointers are stored to facilitate the reconstruction of the best tree.

These backpointers are shown for a partially-parsed sample sentence in Figure 2.3. Each node will only have one left-child pointer and one right-child pointer; however, nodes may be *pointed to* by more pointers. This is where syntactic ambiguities arise. In order to find the Viterbi best parse, they must be distinguished from each other by probabilities.

Finally, the c_ϵ best explaining the substring x_ϵ is the root constituent of the best tree, $\hat{\tau}$ — this tree can be constructed by following the backpointers. Thus, the most likely tree is returned.

CKY Algorithm (with Viterbi probabilities)
Models required: θ_{G}, $\theta_{\text{P-Vit(G)}}$, $\pi_{\mathrm{G}\epsilon}$

x_ϵ	Input string with $\lvert x_\epsilon \rvert$ total words
C	The domain of grammar symbols
C_ϵ	The domain of grammar symbols at the root, $C_\epsilon \subseteq C$
P	A 3D array of probabilities, with dimensions $\lvert x_\epsilon \rvert \times \lvert x_\epsilon \rvert \times \lvert C \rvert$
B	A 3D array of pairs of backpointers, with same dimensions as P
P_ϵ	A 1D array of probabilities at the root, with dimension $\lvert C \rvert$

```
initialize all elements of P to 0
for i = 1 .. |x_ε|
  for each unary rule c_η → x_η                          #Valid preterminals
    set P[i,1,j] = P_{θ_P-Vit(G)}(x_η | c_η)

for i = 2 .. |x_ε|                                       #Length of span
  for j = 1 .. |x_ε|-i+1                                 #Start of span
    for k = 1 .. i-1                                     #Partition of span
      for each binary rule c_η → c_η0 c_η1
        if P[j,k,c_η0] ≠ 0 and P[j+k,i-k,c_η1] ≠ 0
          set P[j,i,c_η] = P_{θ_G}(c_η → c_η0 c_η1) * P[j,k,c_η0] * P[j,k,c_η1]
          if P[j,i,c_η] is the maximum for this i,j
            set B[j,i,c_η] to [j,k] and [j+k,i-k]

for c_ε = 1 .. |C|
  set P_ε[c_ε] = P[1,|x_ε|,c_ε] * P_{π_Gε}(c_ε)
find ĉ_ε corresponding to the maximum probability in P_ε
construct τ̂_ε by tracing backpointers from B[1,|x_ε|,ĉ_η]
return τ̂_ε
```

Figure 2.2: Pseudocode for the CKY algorithm, which parses sentences directly using PCFG rules.

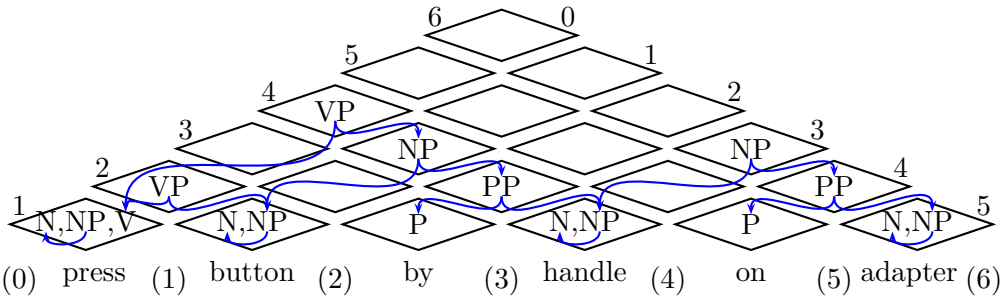


Figure 2.3: Partial CKY parse chart for the utterance '*press button by handle on adaptor*' which is syntactically ambiguous.

This algorithm may be optimized by using grammar or lexicon accessors, memoizing, or a range of other techniques. Regardless, the worst-case runtime complexity is cubic on the number of input words and the number of grammar symbols: $\mathcal{O}(|x_\epsilon|^3|C|^3)$.

2.5 Headword Lexicalization

For each nonterminal grammatical constituent in a tree, one of its children will be the lexical *head* based on syntactic category. Any child node that is not the head is said to be a *modifier*. If such a strategy is used on all nonterminals of a tree, lexical heads will "percolate" up the tree and be called *headwords* (Magerman, 1995). These headwords are a kind of semantics that represent the most important lexical item in a phrase (a word from the subtree of any nonterminal). As such, the nonterminals are *lexicalized*. Figure 2.4 shows a tree lexicalized with headwords.

What does headword lexicalization provide to a parser? Consider the PCFG rules below:

$$\text{VP} \rightarrow \text{VB NN} = .7 \qquad \text{VB} \rightarrow \text{hit} = .5$$
$$\text{VP} \rightarrow \text{VB PP} = .3 \qquad \text{VB} \rightarrow \text{go} = .5$$

There are two completions to a verb phrase (VP), the first of which takes a direct object (NN), the second of which takes a prepositional phrase (PP). The VP rules tell us that having a direct object is more likely, and this would likely be true if the verb (VB) is "hit." However, if the verb is "go," it is unlikely that there would be a direct object, and instead the second rule would be more likely. Thus, the probabilities in the PCFG can be adjusted to consider the main verb in the VP.

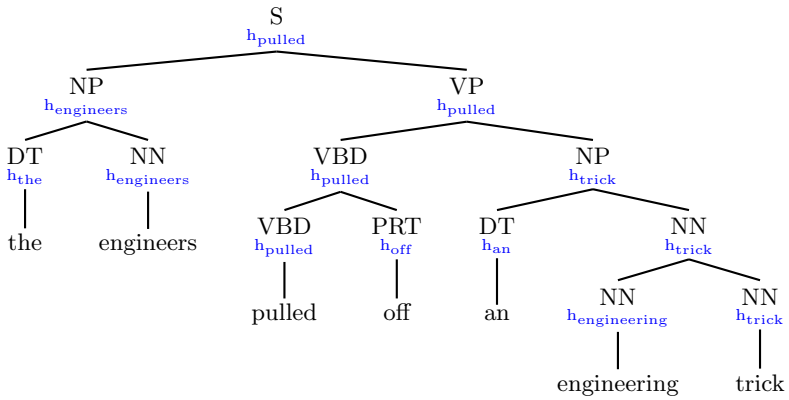


Figure 2.4: A tree in CNF with headwords annotated.

This observation has been used toward parsing with bilexical dependencies (Charniak, 1997; Collins, 1997; Eisner and Satta, 1999), where the headwords of each child in a binary node are considered. High-accuracy parsers are obtainable from lexicalized grammars by head-driven parsing (Collins, 2003), defining useful features for discriminative methods with lexicalization (Charniak, 2000; Charniak and Johnson, 2005), or other strategies.

Chapter 3

Structured Vectorial Semantics

The model of language set forth in this thesis is called *Structured Vectorial Semantics*. Here, 'semantics' does not mean the propositional or predicate logic of formal semantics, but is used more generally to indicate that we will deal with the meaning of utterances. Admittedly, this is still somewhat imprecise, since the framework to be introduced may encompass phenomena from the traditionally separate fields of semantics, pragmatics, and even discourse.

Vectorial accounts of meaning are typically even less precise, in that they do not claim to model traditional linguistic relationships such as syntactic structure or formal logics. The advantage of vector-space models is that they are able to capture latent information in words, and do so in a quantitative manner rather than the symbolic form of logic. These models are usually based on an assumption that a document is just a bag of words, and treat co-occurring words (Lund and Burgess, 1996) as similar in topic and meaning. Thus, techniques like Latent Semantic Analysis (Deerwester et al., 1990; Landauer and Dumais, 1997) or Latent Dirichlet Allocation (Blei, Ng, and Jordan, 2003) implicitly model meaning across traditional linguistic boundaries — including even syntax.

A grammatically-'structured' vectorial framework is advocated for here. In contrast to the bag-of-words assumption, semantic vectors interact significantly with syntactic structure. Also, semantic components are not defined in isolation, but in relation to other semantic components, allowing for context-dependent vector composition.

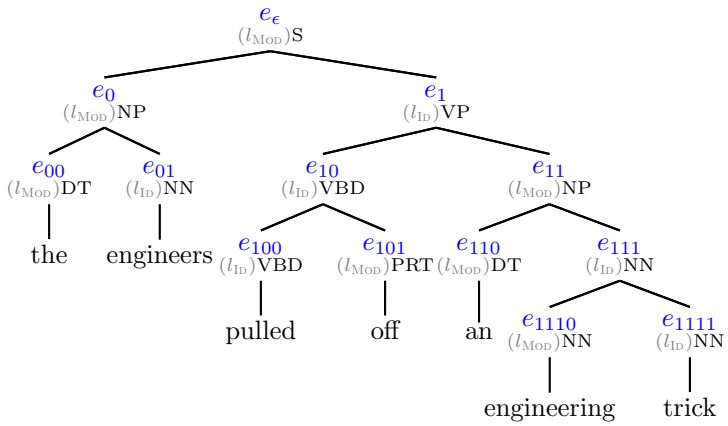


Figure 3.1: Syntax and semantics annotated on a tree. Referents e are subscripted with the node's address. Relations l and syntactic categories c are specific to the example; the relation MOD indicates that the NP is a modifier of S, and ID shows that the VP is the main predicate (head) of S.

Internally, structured vectorial semantics has separate models for syntax and semantics, yet weighs both together when considering the salience of a candidate hypothesis. These models are combined during sentence recognition as a chain of linear operators via matrix multiplication. This chapter will extend the parsing of Section 2.3 to incorporate this structured vectorial semantics.

3.1 Syntactic and Semantic Dependencies

In grammatically-structured semantics, both semantic *concepts* (written e) and *relations* (written l) between these concepts are tied to grammatical structure. The task, then, is to *jointly* determine which grammatical structures and semantic representations best match the observed input.

Figure 3.1 shows an example of a tree that has concepts e, relations l, and syntactic categories c annotated. A probabilistic grammar G producing this figure could feasibly have a rule such as:

$$(l_{\text{MOD}})\text{NP}{:}e_0 \rightarrow (l_{\text{MOD}})\text{DT}{:}e_{00} \;\; (l_{\text{ID}})\text{NN}{:}e_{01}$$

with probability according to a grammar model, θ_{G}, for each concept e_0 in a domain of concepts E. Writing these as variables, this rule would have a probability

$\mathsf{P}_{\theta_{\mathrm{G}}}(lce_{\eta} \rightarrow lce_{\eta 0}\ lce_{\eta 1})$, similar to the rules in Sections 2.2–2.3.

Instead of directly generating joint *lce* probabilities on the right-hand side above, such a syntactic–semantic grammar rule can be factored into (loosely) a syntactic component and a semantic component.

Syntactic–semantic Grammar rule factorization

$$\mathsf{P}_{\theta_{\mathrm{G}}}(lce_{\eta} \rightarrow lce_{\eta 0}\ lce_{\eta 1}) \stackrel{\text{def}}{=} \underbrace{\mathsf{P}_{\theta_{\mathrm{M}}}(lce_{\eta} \rightarrow lc_{\eta 0}\ lc_{\eta 1})}_{\text{syntax}} \cdot \underbrace{\mathsf{P}_{\theta_{\mathrm{L}}}(e_{\eta 0} \,|\, e_{\eta}; l_{\eta 0})}_{\text{left-child semantics}} \cdot \underbrace{\mathsf{P}_{\theta_{\mathrm{L}}}(e_{\eta 1} \,|\, e_{\eta}; l_{\eta 1})}_{\text{right-child semantics}} \quad (3.1)$$

The semicolon in $\mathsf{P}_{\theta_{\mathrm{L}}}(e_{\eta\iota} \,|\, e_{\eta}; l_{\eta\iota})$ is used to show that the relationship is between parent and child concepts, but that the probability is parameterized by (and therefore conditioned on) $l_{\eta\iota}$.

In the θ_{M} model, concepts constrain the generation of syntax and of acceptable relations for a child constituent. In the θ_{L} models, child concepts are probabilistically connected to parent concepts on the basis of the θ_{M}-generated child relations.

Then, an example syntactic rule that would take the place of a joint *lce* dependency might take the form:

$$(l_{\textsc{Mod}})\mathrm{NP}{:}e_0 \rightarrow (l_{\textsc{Mod}})\mathrm{DT}\ \ (l_{\textsc{Id}})\mathrm{NN}$$

with probability according to θ_{M}. The left-child concept e_{00} would be generated from e_0 according to the relation $l_{\textsc{Mod}}$ with some probability, and the right child e_{01} would be similarly generated according to $l_{\textsc{Id}}$.

Viterbi probabilities from Equation 2.6 are extended by marginalizing out child concepts $e_{\eta 0}$ and $e_{\eta 1}$.

Syntactic–semantic Viterbi Probability

$$\mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_{\eta} \,|\, lce_{\eta}) = \max_{\substack{x_{\eta 0} lc_{\eta 0} \\ x_{\eta 1} lc_{\eta 1}}} \sum_{\substack{e_{\eta 0} \\ e_{\eta 1}}} \mathsf{P}_{\theta_{\mathrm{G}}}(lce_{\eta} \rightarrow lce_{\eta 0}\ lce_{\eta 1}) \cdot \mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_{\eta 0} \,|\, lce_{\eta 0}) \cdot \mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_{\eta 1} \,|\, lce_{\eta 1}) \quad (3.2)$$

A possible alternative would be to take the maximum over $e_{\eta 0}$ and $e_{\eta 1}$, either is justified if we adhere to a compositional account of meaning. Grammar rule probabilities from θ_{G} are factored as in Equation 3.1.

Then, given a prior probability $\mathsf{P}_{\pi_{\mathrm{G}\epsilon}}(lce_{\epsilon})$, the probability at the root is:

Syntactic–semantic Root Probability

$$\mathsf{P}(x\, lc_{\epsilon}) = \sum_{e_{\epsilon}} \mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_{\epsilon} \,|\, lce_{\epsilon}) \cdot \mathsf{P}_{\pi_{\mathrm{G}\epsilon}}(lce_{\epsilon}) \tag{3.3}$$

where again, the maximum root probability has a Viterbi probability with corresponding most likely tree $\hat{\tau}_{\epsilon}$.

One obvious reason to factor into θ_{M} and θ_{L} is to reduce the sparsity of training data: syntactic grammars have potentially $|C|^3$ rules, whereas unfactored structured semantic rules could have $(|C||L||E|)^3$ rules.

But the crucial point is that models without this factorization (Matsuzaki, Miyao, and Tsujii, 2005; Petrov et al., 2006) will be unable to maintain a cohesive semantic representation across all syntactic constituents. Such strategies have proven useful for parsing, where the only output of concern is a parse tree (or qualitative analyses of the state-split constituents). However, all other uses for vector-space semantics, such as judging document (or sentence, or phrase, or word) similarity, are precluded from non-cohesive models.

3.2 Vectors, Probabilities, and Denotations

A mathematical framework is now needed by which grammatically-structured semantics can be cast as vectors. In other work, matrices have been used as conditional probability tables, representing spaces like the hidden state space in a HMM transition matrix or grammar symbols in a matrix of grammatical continuations. For structured vectorial semantics, a vector or matrix will represent probabilities in a semantic space.

Notation and definitions are provided here for encapsulating probability distributions *in* vectors, and for defining probability distributions *over* those vectors. This section builds on the linear algebra background of Section 2.1, but much of the notation will be repeated for convenience.

Symbol	Description
Syntactic Model (Sections 2.2–2.3)	
c	Syntactic category, e.g., S, NP, VP, from a domain C
x	String yields, i.e., observed text or concatenated words from a domain X
τ	Tree or subtree
η, ι, κ	(subscripted) A string of '0's and '1's representing the path from a (sub)tree to a descendent
θ_{G}	Conditional probability distribution corresponding to context-free grammar rule expansions
$\theta_{\mathrm{Vit(G)}}$	Probability model for the most likely rules to produce subsumed yields
$\theta_{\mathrm{P\text{-}Vit(G)}}$	Probability model for mapping syntactic constituents to individual words
$\pi_{\mathrm{G}\epsilon}$	Prior probability of root symbol
π_{G}	Prior probability of any symbol
Syntactic–Semantic Model (Section 3.1)	
e_η	Concept variable from the domain E of possible concepts
l_η	Relation symbol for the relationship between concepts
θ_{M}	Syntactic probabilities factored from semantic θ_{G}
θ_{L}	Concept generation probabilities factored from semantic θ_{G}
Probability Vectors (Section 3.2)	
$\mathbf{a}^T$	Row vector representing a prior probability distribution
$e_\eta \mapsto \cdots$	Element-by-element definition of $\mathbf{e}_\eta$
$[\![\phi]\!]$	Indicator function, equal to 1 if ϕ is true, and 0 if ϕ is false
Structured Vectorial Semantics (Section 3.3)	
$\mathbf{M}(\cdots)$	Diagonal matrix representing a grammar rule, conditioned on a parent e, encapsulating θ_{M}
$\mathbf{L}_{\eta\times\iota}$	Relation matrix describing the relationship between concepts at η and ι, encapsulating θ_{L}
$\mathbf{e}_\eta$	Semantic vector containing likelihood probabilities associated with a grammatical constituent
e_i	Constant concept, indexing vectors & matrices
e_η	Variable over the indices of semantic vector $\mathbf{e}_\eta$ (see above)
$\mathbf{e}[\mathrm{e}_i]$	Value (probability) at a particular index in vector $\mathbf{e}$
$\mathrm{d}(\cdot)$	Operation that lists the elements a vector on the diagonal of a square matrix
$\mathbf{1}$	Column vector in which each element is equal to 1
$\mathbf{n}$	Possibly non-linear operator, generalizing $\mathbf{1}$ (Section 5.2)

Table 3.1: Structured vectorial semantics notation.

This paper will use boldfaced uppercase letters to indicate matrices (e.g., $\mathbf{L}$), boldfaced lowercase letters to indicate vectors (e.g., $\mathbf{e}$), and no boldface to indicate any single-valued variable (e.g. l). Often, these variables will technically be functions with arguments written in parentheses, producing the expected vectors or matrices (e.g., $\mathbf{L}(l_\eta)$ will produce some matrix based on the argument l_η). Also, indices of vectors and matrices are constants in upright font (e.g., $\mathrm{e}_1, \ldots, \mathrm{e}_{|\mathbf{e}|}$), variables over those indices are single-value variables in scripted font (e.g., e_η). The contents of vectors and matrices can also be accessed by index (e.g., $\mathbf{e}[\mathrm{e}_1]$ for a constant, $\mathbf{e}[e]$ for a variable).

Again, vectors and matrices will be used to encapsulate probability distributions. This article will adopt the convention that column components of matrices or vectors

represent values of conditioned semantic variables, and row components represent values of modeled semantic variables. The length of vectors is just the domain size $|E|$. Thus, likelihood functions $\mathsf{P}(x \,|\, e)$ are naturally represented as column vectors $\mathbf{e}$. Similarly, prior distributions $\mathsf{P}(e)$ are naturally represented as row (or transpose) vectors $\mathbf{a}^T$, and posterior probabilities $\mathsf{P}(x)$ are vector products $\mathbf{a}^T \cdot \mathbf{e}$.

$$\mathbf{a}^T \cdot \mathbf{e} = \begin{bmatrix} \mathsf{P}(\mathrm{e}_1) & \mathsf{P}(\mathrm{e}_2) & \cdots \end{bmatrix} \cdot \begin{bmatrix} \mathsf{P}(x \,|\, \mathrm{e}_1) \\ \mathsf{P}(x \,|\, \mathrm{e}_2) \\ \vdots \end{bmatrix} \tag{3.4}$$

$$= \sum_i \mathsf{P}(\mathrm{e}_i) \cdot \mathsf{P}(x \,|\, \mathrm{e}_i) \tag{3.5}$$

The indices of $\mathbf{a}^T$ are written as e_i because the codomain (column indices) of $\mathbf{a}^T$ must be the same as the domain (row indices) of $\mathbf{e}$; this will also be true for the other linear algebra operations presented here.

Matrices similarly represent conditional distributions $\mathsf{P}(e_\iota \,|\, e_\eta)$, with the intuition that the rows represent the conditioned variable e_η, and columns represent the modeled variable e_ι. Thus, the probabilistic relationships between all $|E|^2$ concept combinations are considered. Matrix–vector multiplication gives the expected probabilistic interpretation:

$$\mathbf{L} \cdot \mathbf{e} = \begin{bmatrix} \mathsf{P}(\mathrm{e}_1 \,|\, \mathrm{e}_1) & \mathsf{P}(\mathrm{e}_2 \,|\, \mathrm{e}_1) & \cdots \\ \mathsf{P}(\mathrm{e}_1 \,|\, \mathrm{e}_2) & \mathsf{P}(\mathrm{e}_2 \,|\, \mathrm{e}_2) & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \cdot \begin{bmatrix} \mathsf{P}(x \,|\, \mathrm{e}_1) \\ \mathsf{P}(x \,|\, \mathrm{e}_2) \\ \vdots \end{bmatrix} \tag{3.6}$$

$$= \sum_j \mathsf{P}(\mathrm{e}_j \,|\, \mathrm{e}_i) \cdot \mathsf{P}(x \,|\, \mathrm{e}_j) \quad = \quad \mathsf{P}(x \,|\, \mathrm{e}_i) \tag{3.7}$$

The product of two probability distributions that are conditioned on the same variable can be carried out through an operation $\mathrm{d}(\cdot)$, which lists the elements of a column vector on the diagonal of a diagonal matrix. As an example of this diagonal-listing operation:

$$\mathbf{e} = \begin{matrix} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_4 \end{matrix} \overset{\textit{truth}}{\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}} \qquad \mathrm{d}(\mathbf{e}_\eta) = \begin{matrix} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_4 \end{matrix} \overset{\begin{matrix} \textit{codomain} \\ \mathrm{e}_1 \quad \mathrm{e}_2 \quad \mathrm{e}_3 \quad \mathrm{e}_4 \end{matrix}}{\begin{bmatrix} p_1 & 0 & 0 & 0 \\ 0 & p_2 & 0 & 0 \\ 0 & 0 & p_3 & 0 \\ 0 & 0 & 0 & p_4 \end{bmatrix}} \tag{3.8}$$

where the codomain of $\mathrm{d}(\mathbf{e})$ is obviously the same as its domain. Thus, any matrix with compatible domains, multiplied by $\mathrm{d}(\mathbf{e})$, will result in a reweighting of the elements of that matrix.

As a special case of this reweighting, consider a column vector of all ones, $\mathbf{1}$. For any vector $\mathbf{e}$, the $\mathrm{d}(\cdot)$ operation and multiplying by $\mathbf{1}$ are inverse operations: $\mathrm{d}(\mathbf{e}) \cdot \mathbf{1} = \mathbf{e}$. Also, for two vectors $\mathbf{e}_\eta$ and $\mathbf{e}_\iota$, the product $\mathrm{d}(\mathbf{e}_\eta) \cdot \mathrm{d}(\mathbf{e}_\iota) \cdot \mathbf{1}$ is just the point-wise product of $\mathbf{e}_\eta$ and $\mathbf{e}_\iota$.

Probability-encapsulating vectors can themselves be incorporated into other probability distributions. Distributions modeling these vectors and matrices are deterministic, equal to one if the vector or matrix contains the correct probabilities, zero otherwise. This is done via an indicator function $[\![\cdot]\!]$, where $[\![\phi]\!] = 1$ if ϕ is true, 0 otherwise. For example, a vectorized prior probability distribution would be:

$$\mathsf{P}(\mathbf{a}_\eta^T) = [\![\mathbf{a}_\eta^T = e_\eta \mapsto \mathsf{P}(e_\eta)]\!] \tag{3.9}$$

where $e_\eta \mapsto \cdots$ indicates that what follows it is an element-by-element definition of $\mathbf{e}_\eta$.

Both the probability notation and the linear algebra notation will be used to define structured vectorial semantics.

3.3 Structured Vectorial Semantic Parsing

The probabilistic semantic recognizer described in Section 3.1 can now be encapsulated in linear algebraic notation. General vector and probability definitions are provided here, with examples delayed until a specific semantic space is considered in Section 3.4.

First, per-concept Viterbi probabilities in a syntactic context lc_η are encapsulated as column vectors:

$$\mathbf{e}_\eta = e_\eta \mapsto \mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x \,|\, e_\eta; lc_\eta) \tag{3.10}$$

Prior probability distributions for each concept are row vectors:

$$\mathbf{a}_\eta^T = e_\eta \mapsto \mathsf{P}_{\pi_{\mathrm{G}}}(lce_\eta) \tag{3.11}$$

Additionally, the syntactic probabilities θ_{M} are captured in a matrix (a diagonally-listed vector), with a different probability for each possible concept:

$$\mathbf{m}(lc_\eta \to lc_{\eta 0}\ lc_{\eta 1}) \stackrel{\text{def}}{=} e_\eta \mapsto \mathsf{P}_{\theta_{\mathrm{M}}}(lce_\eta \to lc_{\eta 0}\ lc_{\eta 1}) \tag{3.12}$$

$$\mathbf{M}(lc_\eta \to lc_{\eta 0}\ lc_{\eta 1}) = \mathrm{d}(\mathbf{m}(lc_\eta \to lc_{\eta 0}\ lc_{\eta 1})) \tag{3.13}$$

Next, relation matrices are functions from concepts of some constituent η to concepts of a child $\eta\iota$ with some probability:

$$\mathbf{L}(l_{\eta\iota}) = \mathbf{L}_{\eta\times\eta\iota}(l_{\eta\iota}) \stackrel{\text{def}}{=} e_\eta \mapsto e_{\eta\iota} \mapsto \mathsf{P}_{\theta_{\mathrm{L}}}(e_{\eta\iota} \,|\, e_\eta; l_{\eta\iota}) \tag{3.14}$$

Combining Equation 3.10 with these vector and matrix encapsulations of probability distributions, parsing probabilities can be encapsulated in a vector:

$$\mathbf{e}_\eta = e_\eta \mapsto \max_{\substack{xlc_{\eta 0}\\ xlc_{\eta 1}}} \sum_{\substack{e_{\eta 0}\\ e_{\eta 1}}} \mathsf{P}_{\theta_{\mathrm{G}}}(lce_\eta \to lce_{\eta 0}\ lce_{\eta 1}) \cdot \mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_{\eta 0} \,|\, lce_{\eta 0}) \cdot \mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_{\eta 1} \,|\, lce_{\eta 1}) \tag{3.15}$$

$$= e_\eta \mapsto \max_{\substack{xlc_{\eta 0}\\ xlc_{\eta 1}}} \sum_{\substack{e_{\eta 0}\\ e_{\eta 1}}} \mathsf{P}_{\theta_{\mathrm{M}}}(lce_\eta \to lc_{\eta 0}\ lc_{\eta 1}) \cdot \mathsf{P}_{\theta_{\mathrm{L}}}(e_{\eta 0} \,|\, e_\eta; l_{\eta 0}) \cdot \mathsf{P}_{\theta_{\mathrm{L}}}(e_{\eta 1} \,|\, e_\eta; l_{\eta 1}) \cdot \mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_{\eta 0} \,|\, lce_{\eta 0}) \cdot \mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_{\eta 1} \,|\, lce_{\eta 1}) \tag{3.16}$$

$$= e_\eta \mapsto \max_{\substack{xlc_{\eta 0}\\ xlc_{\eta 1}}} \mathsf{P}_{\theta_{\mathrm{M}}}(lce_\eta \to lc_{\eta 0}\ lc_{\eta 1}) \cdot \sum_{e_{\eta 0}} \mathsf{P}_{\theta_{\mathrm{L}}}(e_{\eta 0} \,|\, e_\eta; l_{\eta 0}) \cdot \mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_{\eta 0} \,|\, lce_{\eta 0}) \cdot \sum_{e_{\eta 1}} \mathsf{P}_{\theta_{\mathrm{L}}}(e_{\eta 1} \,|\, e_\eta; l_{\eta 1}) \cdot \mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_{\eta 1} \,|\, lce_{\eta 1}) \tag{3.17}$$

$$= \mathbf{M}(lc_\eta \to lc_{\eta 0}\ lc_{\eta 1}) \ \cdot\ \mathrm{d}(\mathbf{L}_{\eta\times\eta 0}(l_{\eta 0}) \cdot \mathbf{e}_{\eta 0}) \ \cdot\ \mathrm{d}(\mathbf{L}_{\eta\times\eta 1}(l_{\eta 1}) \cdot \mathbf{e}_{\eta 1}) \ \cdot\ \mathbf{1} \tag{3.18}$$

The syntactic–semantic grammar rule factorization is first applied; the result is rearranged to show how it is equivalent to a normal matrix product. Note that $\mathbf{1}$ is the default, linear way to return a vector at the end of the last equation. A more general definition of matrix-to-vector functions, $\mathbf{n}$, would include other (possibly nonlinear) operators. Also, since this already incorporates the factorization of Equation 3.1, no vectorized version of θ_{G} is necessary.

The vector alone does not include the syntactic information, though. So we need to define a (deterministic) Viterbi probability over this vector and its relevant syntactic

variables:

$$\mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_\eta \,|\, lc\mathbf{e}_\eta) =$$
$$[\![\ \mathbf{e}_\eta = \ \mathbf{M}(lc_\eta \to lc_{\eta 0}\ lc_{\eta 1}) \cdot \mathrm{d}(\mathbf{L}_{\eta\times\eta 0}(l_{\eta 0}) \cdot \mathbf{e}_{\eta 0}) \cdot \mathrm{d}(\mathbf{L}_{\eta\times\eta 1}(l_{\eta 1}) \cdot \mathbf{e}_{\eta 1}) \cdot \mathbf{1}\ ]\!] \quad (3.19)$$

The deterministic probability ensures that the syntax and yield match the calculated semantic vector $\mathbf{e}$.

Given a prior distribution at the start symbol of the grammar,

$$\mathsf{P}_{\pi_{\mathrm{G}\epsilon}}(lc\mathbf{a}_\eta^T) = [\![\mathbf{a}_\eta^T = e_\eta \mapsto \mathsf{P}_{\pi_{\mathrm{G}\epsilon}}(lce_\eta)]\!] \quad (3.20)$$

the probability at the root is:

SVS Root Probability

$$\mathsf{P}(x\ lc_\epsilon) = \mathbf{a}_\epsilon^T \cdot \mathbf{e}_\epsilon \ \cdot\ \mathsf{P}_{\pi_{\mathrm{G}\epsilon}}(lc\mathbf{a}_\epsilon^T) \cdot \mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x \,|\, lc\mathbf{e}_\epsilon) \quad (3.21)$$

where the first two terms give the encapsulated probabilities, and the last two terms deterministically ensure that the indicated tree τ_η is indeed the one generated. As before, the most likely tree $\hat{\tau}$ is the tree that maximizes this probability, and can be constructed from the best child trees.

A sequence of linear operations (vector and matrix multiplication) ostensibly represents semantic probabilities; however, defining probabilities over semantic vectors as in Equations 3.19 and 3.20 allows them to keep track of syntactic constituents. In this way, syntactic θ_{M} can be weighed together in semantic vectors. Thus, after a series of deterministic vectorized probability distributions, the encapsulated probabilities are "unlocked" at the root node.

It should be apparent that the semantics of the most likely tree, $\mathbf{e}_{\hat{\tau}_\epsilon}$, have a distinct likelihood for each e in $|E|$. This vector at the root node can be construed as the composed vectorial semantic information for the whole parsed sentence. This semantic information may be used in downstream processes or applications that require an analysis of quantitative semantic information in text. Similar vectors can be obtained anywhere on the best tree, or even any best subtree.

In this formulation, probability definitions from Section 3.1 have been completely preserved. However, close scrutiny of Equation 3.15 indicates that this definition causes

reconstruction of the most likely tree to be difficult in a vectorized parser. Since the maximum is taken over $x\, lc_{\eta 0}$ and $x\, lc_{\eta 1}$ for each index e_η, each element could have its maximum probability from different rules $lce_\eta \to lce_{\eta 0}\ lce_{\eta 1}$.

An approximation can be made that does not requires extra machinery alongside the vectors in order to find the best tree. The key is to take the maximum of whole vectors rather than of vector elements. Redefining a vector for some hypothesized grammar rule $lc_\eta \to lc_{\eta 0}\ lc_{\eta 1}$:

SVS Composition Equation

$$\mathbf{e}_\eta \stackrel{\text{def}}{=} e_\eta \mapsto \sum_{\substack{e_{\eta 0} \\ e_{\eta 1}}} \mathsf{P}_{\theta_{\mathrm{G}}}(lce_\eta \to lce_{\eta 0}\ lce_{\eta 1}) \cdot \mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_{\eta 0} \,|\, lce_{\eta 0}) \cdot \mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_{\eta 1} \,|\, lce_{\eta 1}) \tag{3.22}$$

which leaves out the maximization operation. The approximated Viterbi probability picks the vector that maximizes the joint probability at a particular node, similar to the maximum operation at the root. As at the root, a prior probability distribution is needed; however, this differs from the root probability ($\mathsf{P}_{\pi_{\mathrm{G}\epsilon}}(lc\mathbf{a}_\epsilon^T)$) in that it should consider constituents that could be at any node ($\mathsf{P}_{\pi_{\mathrm{G}}}(lc\mathbf{a}_\eta^T) = [\![\mathbf{a}_\eta^T = e_\eta \mapsto \mathsf{P}_{\pi_{\mathrm{G}}}(lce_\eta)]\!]$):

SVS Viterbi Probability

$$\mathsf{P}_{\theta_{\mathrm{Vit(G)}}}(x_\eta \,|\, lc\mathbf{e}_\eta) \stackrel{\text{def}}{=} [\![\ \mathbf{e}_\eta = \arg\max_{lc\mathbf{e}_\kappa}\ \mathbf{a}_\kappa^T \mathbf{e}_\kappa\ ]\!] \tag{3.23}$$

This approximation is tested for empirical soundness in Section 3.5. It will be shown to be a harmless approximation; so rather than directly using the vector definitions of Equations 3.15–3.19, we will use the same notation of $\mathbf{e}$ to always refer to this approximate version. Thus, SVS may then be considered a vectorial framework both theoretically and implementationally.

The result is a flexible, learnable semantic model that can be extended to a variety of semantic spaces and tasks.

3.4 Bilexical Parsing with SVS

In lexicalized parsing (Collins, 1996; Collins, 1997), syntactic categories are augmented with headword dependencies h. As a degenerate case, the semantic space for structured vectorial semantics can correspond to these headwords, i.e., $e \stackrel{\text{def}}{=} h$ and relations can be restricted to either heads or modifiers, $l \in \{l_{\text{ID}}, l_{\text{MOD}}\}$. With lexical dependencies on both children, this instantiation of SVS effectively accomplishes bilexical parsing (Charniak, 1997; Collins, 1997; Eisner and Satta, 1999).

Though headwords are usually considered a syntactic feature, they can be considered a form of degenerate semantics: headwords are concepts that summarize the information below them; head and modifier relations are dependencies between these concepts. This particular semantic space definition is important because it establishes a basis for empirical comparison of structured vectorial semantics with standard broad-coverage parsing models. Chapters 4 and 5 give more plausible models of semantics.

To exemplify structured vectorial semantic parsing with headwords, a running example corresponding to the best parse of the first two words of Figure 3.1 ("the engineers") is given in this section. This involves calculating the Viterbi probability:

$$\mathsf{P}_{\theta_{\text{Vit(G)}}}(x_0 \,|\, lc\mathbf{e}_0) = \\ [\![\ \mathbf{e}_0 = \ \mathbf{M}(lc_0 \rightarrow lc_{00}\ lc_{01}) \cdot \text{d}(\mathbf{L}_{0\times 00}(l_{00}) \cdot \mathbf{e}_{00}) \cdot \text{d}(\mathbf{L}_{0\times 01}(l_{01}) \cdot \mathbf{e}_{01}) \cdot \mathbf{1}\ ]\!]$$

Let us assume that the space of concepts is $E = H = \{\text{h}_{\text{pulled}}, \text{h}_{\text{the}}, \text{h}_{\text{unk}}\}$, abbreviated as $\{\text{h}_\text{p}, \text{h}_\text{t}, \text{h}_\text{u}\}$ where the last concept is a constant for infrequently-observed words.

Vectors are indexed by individual lexical items, and express the conditional probability that each lexical item will produce an observed yield in syntactic context. So at the level of word generation, the left child concept will be assigned from $\mathsf{P}_{\theta_{\text{P-Vit(G)}}}(\text{the} \,|\, l_{\text{MOD}}{:}\text{DT}\{\text{h}_\text{t}\})$, and the right child will be assigned from $\mathsf{P}_{\theta_{\text{P-Vit(G)}}}(\text{engineers} \,|\, l_{\text{ID}}{:}\,\text{NP}\{\text{h}_\text{u}\})$:

$$\mathbf{e}_{\eta 0} = \begin{matrix} \\ \textit{headwords} \\ \text{h}_\text{u} \\ \text{h}_\text{t} \\ \text{h}_\text{p} \end{matrix} \overset{\textit{truth}}{\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}} \qquad \mathbf{e}_{\eta 1} = \begin{matrix} \\ \textit{headwords} \\ \text{h}_\text{u} \\ \text{h}_\text{t} \\ \text{h}_\text{p} \end{matrix} \overset{\textit{truth}}{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}} \tag{3.24}$$

More generally, preterminal probabilities in headword-lexicalization SVS are:

$$\mathsf{P}_{\theta_{\text{P-Vit(G)}}}(x_\eta \,|\, lch_\eta) = \begin{cases} \tilde{\mathsf{P}}(x_\eta \,|\, lch_\eta) & x_\eta \in H \\ \mathsf{P}_{\theta_{\text{P-Vit(G)}}}(x_\eta \,|\, c_\eta) & x_\eta \notin H \end{cases} \tag{3.25}$$

where the first case is an empirical model that can be learned from trees, and the second is a backoff model learned by some other method.

Grammar probabilities are encapsulated per-concept in the $\mathbf{M}$ matrix. The whole binary rule identifies which diagonal matrix to use:

$$\mathbf{M}(l_{\text{MOD}}{:}\text{NP} \rightarrow l_{\text{MOD}}{:}\text{DT} \;\; l_{\text{ID}}{:}\text{NN}) = \begin{array}{c} \\ \textit{parent } \eta \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccc} \textit{parent } \eta & & \\ \text{h}_\text{p} & \text{h}_\text{t} & \text{h}_\text{u} \end{array} \\ \begin{bmatrix} .2 & 0 & 0 \\ 0 & .1 & 0 \\ 0 & 0 & .4 \end{bmatrix} \end{array} \begin{array}{l} \\ \text{h}_\text{p} \\ \text{h}_\text{t} \\ \text{h}_\text{u} \end{array} \tag{3.26}$$

Relation matrices $\mathbf{L}$ are functions from headwords h_η to child headwords $h_{\eta 0}$ or $h_{\eta 1}$, with some probability. With these two relations, this framework easily captures standard bilexical dependencies. A nonterminal constituent will share its headword with one of its sub-constituents (its head), so this sub-constituent will be associated with the identity matrix $\mathbf{L}_{\text{ID}}$ as its labeled role relation. The other sub-constituent (a modifier) will then be associated with a generalized modifier relation $\mathbf{L}_{\text{MOD}}$, or one of a set of argument-specific relations. In our example, the relation l_{ID} produces $\mathbf{L}(l_{\text{ID}}) = \mathbf{I}$, a 3×3 identity matrix; the modifier relation may plausibly be:

$$\mathbf{L}_{\text{MOD}}(l_{00}) = \begin{array}{c} \\ \textit{parent } 0 \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccc} \textit{child } 00 & & \\ \text{h}_\text{p} & \text{h}_\text{t} & \text{h}_\text{u} \end{array} \\ \begin{bmatrix} 0 & .2 & .8 \\ 0 & 0 & 0 \\ .3 & .7 & 0 \end{bmatrix} \end{array} \begin{array}{l} \\ \text{h}_\text{p} \\ \text{h}_\text{t} \\ \text{h}_\text{u} \end{array} \qquad \mathbf{L}_{\text{ID}}(l_{01}) = \begin{array}{c} \\ \textit{parent } 0 \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccc} \textit{child } 01 & & \\ \text{h}_\text{p} & \text{h}_\text{t} & \text{h}_\text{u} \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array} \begin{array}{l} \\ \text{h}_\text{p} \\ \text{h}_\text{t} \\ \text{h}_\text{u} \end{array} = \mathbf{I} \tag{3.27}$$

In some bilexical parsers, the lexical head may be conditioned on phrasal categories — or other local factors such as whether the child is before or after the head. This can be done in structured vectorial semantics by diversifying the set of labeled relations: e.g., $\mathbf{L}(l_{\text{MOD,DT,NN}})$.

The encapsulating vector is then put into the deterministic probability distribution:

$$\mathbf{e}_0 = \begin{bmatrix} .2 & 0 & 0 \\ 0 & .1 & 0 \\ 0 & 0 & .4 \end{bmatrix} \text{d}\left(\begin{bmatrix} 0 & .2 & .8 \\ 0 & 0 & 0 \\ .3 & .7 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right) \cdot \text{d}\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{3.28}$$

$$\mathsf{P}_{\theta_{\text{Vit(G)}}}(x_\eta \,|\, lc\mathbf{e}_\eta) = [\![\mathbf{e}_0 = \begin{matrix} \\ \mathrm{h_p} \\ \mathrm{h_t} \\ \mathrm{h_u} \end{matrix} \overset{truth}{\begin{bmatrix} 0 \\ 0 \\ 0.028 \end{bmatrix}}]\!] \tag{3.29}$$

Notice that there is only one non-zero element in the result, corresponding to the correct headword of NP (if this grammar rule is used). Non-zero probabilities will only appear where the concepts are consistent with headword percolation in the hypothesized tree; thus, headword-lexicalization with SVS adds no dependencies beyond normal lexicalization.

Note that this differs from other methods of lexicalization due to the consistent factorization of θ_{G} into θ_{M} and θ_{L} — to exclusively use this factorization is potentially detrimental, since other accounts of lexicalization use a myriad of other dependencies. This factorization is crucial, though, because without it, structured vectorial semantics would not have a plausible or tractable way to compose concepts. Therefore, the following section evaluates the performance of this θ_{M}- and θ_{L}-based model on a broad-coverage lexicalized parsing task.

3.5 Empirical Validation of SVS Assumptions

Evaluations were carried out on the standard Wall Street Journal parsing task to empirically verify the assumptions and the parsing implementation for structured vectorial semantics.

Sections 02-21 were used as training data. Trees were binarized; then, each branch was annotated with a head relation l_{ID} or a modifier relation l_{MOD} according to a binarized version of headword percolation rules (Magerman, 1995; Collins, 1997); the parent headword was propagated up from its head constituent. Only the most common post-propagation headwords (e.g., $\mathrm{h}_1, \ldots, \mathrm{h}_{50}$) were stored, and the rest were assigned a constant, 'unknown' headword category.

From counts on the binary rules of these annotated trees, PCFGs according to θ_{M} and θ_{L} were estimated, and preterminal probabilities $\theta_{\text{P-Vit(G)}}$ with full dependencies were also found. Modifier relations l_{MOD} were deterministically augmented with their syntactic context. Both c and l symbols appearing fewer than 10 times in the whole corpus were assigned 'unknown' categories as well.

Additionally, the syntactic model θ_{M} was conditioned on the center-embedding depth d of the expanding constituent and whether the expanding constituent was a left or right child. This last transformation has been shown to have little effect on parsing performance (Schuler et al., 2010), and sets the stage for future, incremental versions of this parser (to be explained in Part II).

The trained PCFG models were evaluated on WSJ Section 22. For controlled comparison, no backoff was used between a full θ_{G} version and the factored $\theta_{\mathrm{M}}, \theta_{\mathrm{L}}$ version. It should be noted that although concepts and relations are also recognized in the structured semantic bilexical parser, these variables are marginalized out or ignored for the evaluations. In semantic spaces other than headwords, the vectorial information at the root $\mathbf{e}_{\epsilon}$, or at any node $\mathbf{e}_{\eta}$, may be a useful quantity in its own right.

Parsing accuracy evaluations are typically measured by three metrics:

$$\begin{aligned} \textbf{Precision} &= \frac{\text{\# constituents correct, hypothesized}}{\text{total \# constituents, hypothesized}} \\ \textbf{Recall} &= \frac{\text{\# constituents correct, hypothesized}}{\text{total \# constituents, standard}} \\ \textbf{F-score} &= \frac{\text{precision} \cdot \text{recall}}{\text{precision} \times \text{recall}} \end{aligned} \tag{3.30}$$

These are calculated for four parsers, to be discussed below.

Headword-lex.	LR	LP	F
no vec, 00hw	83.14	83.14	83.14
no vec, 50hw	83.78	83.95	83.86
vec, 00hw	83.29	83.35	83.32
vec, 50hw	84.21	84.26	84.23

Table 3.2: Effect of grammar factorization and vectorization, for lexicalized models on sentences shorter than 40 words in WSJ Section 22, with punctuation.

3.5.1 Effect of SVS Grammar Factorization

The first comparison to be noted on Table 3.2 is between the first two lines: a non-vectorized parser trained on trees with only one constant headword against a parser trained on trees with 50 headwords. The rest of the transformations remain the same. The constant-headword case is equivalent to a syntactic grammar with rules

$lc_\eta \rightarrow lc_{\eta 0}\, lc_{\eta 1}$ since θ_{L} is deterministic, thus the comparison measures only the effect of adding lexical (i.e., degenerate semantic) dependencies into the model.

If the factorization into θ_{M} and θ_{L} is a valid one, then the 50 headwords from a non-deterministic θ_{L} will increase the parser's performance similar to other methods of lexicalization. However, if this factorization does not preserve conceptual headword information, it will not improve parsing performance. The test therefore addresses the fundamental question of the *assumptions* in structured vectorial semantics.

Table 3.2 shows that bilexical parsing with 50 headwords according to a structured semantic model does in fact preserve expected gains in lexicalization, with error reduction of 4.27%. These results are similar to lexicalization gains reported by Gildea (2001). This means that under the factorization of Equation 3.1, conceptual information propagating up from observed text can indeed positively influence sentence recognition. Since the semantic space of headwords is a minimal one, other vector definitions, under the same structured semantic model, would be expected to further improve recognition.

3.5.2 Effect of Semantic Vectorization on Accuracy

Section 3.3 made the claim that encapsulating probability distributions in vectors preserves the probabilities in the non-vectorized, syntactic–semantic equations of Section 3.1. However, an approximation had to be made in Equation 3.23 in order to allow for Viterbi trees to be calculated on whole vectors, rather than on individual elements of each vector.

If the probabilities perform the same empirically, these approximate vectorized implementations of the parser may be evaluated instead of non-vectorized (or exact-vector) versions. This would allow single-element vector spaces to easily be defined, which could serve as head-to-head baselines for richer vector spaces.

A comparison of lines 1 & 3 and lines 2 & 4 of Table 3.2 shows that vectorization with approximation produces negligible or positive parsing accuracy differences. Thus, it is possible for a CKY parser to probabilistically decide between whole vectors, rather than separately considering each concept element's best syntactic rules.

Since the process of vectorization itself incurs no penalty, different vectorial semantic spaces can be more freely applied to the parser, as long as these semantic spaces provide probability models for θ_{L} and $\theta_{\text{P-Vit(G)}}$. The syntactic model, θ_{M}, may remain unchanged

or be deterministically re-estimated.

An important use of vectorization is to optimize parsing speed in implementations. Contiguous storage of information leads to more efficient multiplication of relevant probabilities. For example, the non-vectorized 10-headword models parsed WSJ Section 23 in 26:52 hours, whereas the vectorized 10-headword model did it in 9:38 hours.[1] Matrix multiplication is well-studied and eminently parallelizable, and may thus be further optimized in the future. Chapter 4 will quantify this speed optimization in more-difficult dense-matrix multiplication operations.

3.6 Summary

This chapter has introduced Structured Vectorial Semantics, a model of language that fully interleaves vectorial semantic composition with syntactic parsing. The language model was defined on a standard PCFG grammar, and is therefore highly flexible. Semantic concepts and relations between concepts that adhered to grammatical structure were defined, and associated probabilities were encapsulated in vectors and matrices. To formally include syntactic information, deterministic probabilities over the vectors themselves were defined for particular grammatical conditions.

Structured Vectorial Semantics is based on the assumption that (roughly) syntactic and semantic models can be factored as a separately contributing factors at each step of probabilistic recognition. This assumption was evaluated for a minimal case of structured semantics — that of headword concepts and head–modifier relations. Positive results show that even with this factorization, accounting semantics alongside syntax improves overall estimates of phrase structure.

In addition, it was shown that the process of vectorization does not incur any penalties in parsing performance, but may improve implementation speed. Thus, Structured Vectorial Semantics is both theoretically and implementationally amenable to incorporating complex vectorial semantic models into syntactic parsing.

[1] This is much slower than a typical CKY parser with a regular syntactic grammar. The dependence on center-embedding depth and parents as a right- or left-child account for some of this, since it multiplies the number of rules that must be explored. The fact that on-the-fly composition needs to occur (rather than a simple grammar rule look-up) also contributes to the runtime.

Chapter 4

Relational Clustering with SVS

The Structured Vectorial Semantic (SVS) framework presented in Chapter 3 may be instantiated in many ways. One such instantiation was given in Section 3.4, in which concepts were defined as headwords to achieve lexicalized parsing. The probability models in that instantiation were directly determined from automatically-annotated trees. This chapter begins by generalizing the input required for any instantiation of SVS, then defines a particular plausible language model within the SVS framework. Familiarity with Chapter 3, including the lexicalized parsing instantiation of Section 3.4, is assumed for this chapter.

High-accuracy parsers have been built using features derived from lexical dependencies (Collins, 1999; Charniak, 2000; Charniak and Johnson, 2005), but the independent contribution of lexicalization itself is only a small part of these parsing gains (Gildea, 2001). Here, I aim to show that the benefits of lexicalization may be improved by a relational clustering (Taskar, Segal, and Koller, 2001) of headwords into latent, abstract concepts.

As noted in Section 3.4, headword lexicalization can be thought of as a very minimal type of semantics, where the headword represents the most important concept of a phrase. In contrast, vector-space models construe the semantics of a word as a collection of concepts expressed to varying degrees. For example, headword lexicalization would express the meaning of the noun phrase *the big bonfire* with just a headword (*bonfire*); vector-space semantics might indicate that the noun phrase is a combination of a 'things that are hot' concept and a 'things that are larger than a person' concept and a 'things

done for celebratory purposes' concept.[1]

The semantic model here is in the family of dimensionality reduction techniques like Latent Semantic Analysis (Deerwester et al., 1990; Landauer and Dumais, 1997), probabilistic LSA (Hofmann, 1999; Hofmann, 2001), Latent Dirichlet Allocation (Blei, Ng, and Jordan, 2003), and tensor clustering (Banerjee, Basu, and Merugu, 2007). Following the observation that words consistently co-occurring in similar documents will be similar or related to each other (Harris, 1954), these techniques build *co-occurrence matrices* (Lund and Burgess, 1996) that give a quantitative representation for documents or words. Then, they map the high-dimensional space of words to a lower-dimensional space of latent concepts or clusters such that the co-occurrence relationship is preserved, but distilled into its most important parts.

This chapter explores an application of similar dimensionality reduction techniques to produce semantic clusters from headwords co-occurring in dependency (e.g., head or modifier) relations. Used in statistical parsing, conceptual clusters are shown here to yield significant improvements in accuracy over headword lexicalization.

Training these abstract concepts is very similar to automatic state-splitting for unlexicalized grammars (Matsuzaki, Miyao, and Tsujii, 2005; Petrov et al., 2006; Petrov and Klein, 2007). Algorithms like EM are used to automatically estimate latent variables which refine the parsing process in both cases. However, syntactic state-splitting and relational headword clustering approaches have different goals: splitting grammatical states improves syntactic rule expansions for finding syntactic structure; relational clustering draws on propagated lexical information to build a full-fledged semantic component — concepts that maintain a coherent interpretation across all syntactic categories.

Because the relationally-clustered model explores previously untapped semantic information, it may be applied in conjunction with other parsing techniques. PCFG-based unlexicalized methods may include relationally-clustered SVS if they are given headword percolation rules; lexicalized models may define features over clusters instead of (or in conjunction with) headwords; dependency parsers may directly make use of relationally-clustered information without the phrase structure component.

[1] Vector-space models differ in the interpretability of the concepts they produce.

Language Model for Structured Vectorial Semantics

Syntactic model	$\mathsf{P}_{\theta_{\mathrm{M}}}(lce_\eta \rightarrow lc_{\eta 0}\, lc_{\eta 1})$	(4.1)
Semantic model	$\mathsf{P}_{\theta_{\mathrm{L}}}(e_{\eta\iota} \,|\, e_\eta; l_{\eta\iota})$	(4.2)
Preterminal model	$\mathsf{P}_{\theta_{\text{P-Vit(G)}}}(x_\eta \,|\, lce_\eta)$	(4.3)
Root constituent model	$\mathsf{P}_{\pi_{\mathrm{G}\epsilon}}(lce_\epsilon)$	(4.4)
Any constituent model	$\mathsf{P}_{\pi_{\mathrm{G}}}(lce_\eta)$	(4.5)

Figure 4.1: Probability models needed as input for Structured Vectorial Semantics.

The evaluations at the end of this chapter test parameters such as number of clusters, number of headwords, and initial random seed with respect to parsing accuracy for relationally-clustered SVS.

The remainder of this chapter is organized as follows: Section 4.1 gives the probability models that define an instantiated SVS language model; Section 4.2 extends lexicalization to relationally-clustered semantics using EM; Section 4.3 shows how the clustered semantics define the necessary SVS probability models; Section 4.4 describes implementation details and evalutaion results; Section 4.5 draws conclusions, gives comparisons, and points to future prospects.

4.1 Language Models in SVS Instantiations

SVS requires a number of probability models as input, as shown in Figure 4.1. With the exception of the semantic model θ_{L}, these probability models are extensions of those that would be used in a syntactic parser, in that they consider semantic concepts alongside syntactic constituents. To define or estimate these probability models is to instantiate Structured Vectorial Semantics.

Recall that the headword-lexicalized version of SVS could be defined by setting $e \stackrel{\text{def}}{=} h$. Since treebank trees can be fully annotated with headwords h and relations l, the models θ_{M}, θ_{L}, $\pi_{\mathrm{G}\epsilon}$, and π_{G} can be obtained directly from data. Empirical probabilities of this kind are denoted with a tilde. The preterminal probability $\theta_{\text{P-Vit(G)}}$ for headword-lexicalization SVS was defined in Equation 3.25 on page 29.

However, when models θ_{M}, θ_{L}, $\theta_{\text{P-Vit(G)}}$ and accompanying parsing equations (Equations 3.1–3.3, page 20) are defined on abstract concepts, probabilities are no longer empirically available from annotated trees, and must be determined by some other method. This is the interface where vector space models must plug in to Structured Vectorial Semantics.

4.2 Inducing Relational Clusters

A first pass at defining clustered semantic concepts from lexical items might be to use word co-occurrences in documents, as in vector-space models like pLSA or LDA. Each lexical item in observed text could easily be represented by a dimensionality-reduced vector. However, it would be difficult to annotate a tree with abstract concepts based on documents, or to directly propagate these concepts into the SVS language model.

This chapter's solution to these problems is to estimate abstract-concept behavior from training trees deterministically augmented with headwords. Thus, concepts are no longer *defined* as headwords, i.e. $e_\eta \stackrel{\text{def}}{\neq} h_\eta$, but headwords h_η will still be used in defining semantic probabilities. Additionally, co-occurrences are defined with more discretion, specifying co-occurring relationships to be the same as those used with headword-lexicalization SVS. Thus, relational clustering will be defined whose locale of co-occurrences is parent–child headwords under a relation l. The Expectation–Maximization (EM) algorithm (Dempster, Laird, and Rubin, 1977) will be used to learn syntactic–semantic referents and relations that are used to instantiate the parser.

The parameters of three probability models will be hidden variables in EM:

1. $\hat{\mathsf{P}}_{\theta_{\mathrm{M}}}(lce_\eta \rightarrow lc_{\eta 0}\, lc_{\eta 1})$, the concept-dependent syntactic expansion rules
2. $\hat{\mathsf{P}}_{\theta_{\mathrm{L}}}(e_{\eta\iota} \,|\, e_\eta; l_{\eta\iota})$, the probability that a parent concept e_η will produce a child concept $e_{\eta\iota}$ if the associated relation is $l_{\eta\iota}$
3. $\hat{\mathsf{P}}_{\theta_{\mathrm{H}}}(h_{\eta\iota} \,|\, lce_{\eta\iota})$, a probabilistic mapping from clusters to headwords

At maximum, there are $|L|^3|C|^3|E|+|E|^2|L|+|L||C||E||H|$ parameters to estimate, though incompatible l and c values can be ignored in practice. These parameters are randomly initialized in preparation for the Expectation–Maximization algorithm.

The EM algorithm is done in two steps, an Expectation step (E-step) and a Maximization step (M-step).

E-step:

$$\hat{\mathsf{P}}(e_\eta, e_{\eta 0}, e_{\eta 1} \,|\, lc_\eta, lc_{\eta 0}, lc_{\eta 1}) = \frac{\hat{\mathsf{P}}_{\theta_{\text{Out}}}(lce_\eta, \overline{lch_\epsilon} - \overline{lch_\eta}) \cdot \hat{\mathsf{P}}_{\theta_{\text{Ins}}}(\overline{lch_\eta} \,|\, lce_\eta)}{\hat{\mathsf{P}}(\overline{lch_\epsilon})} \tag{4.6}$$

$$\tilde{\mathsf{P}}(lce_\eta, lce_{\eta 0}, lce_{\eta 1}) = \hat{\mathsf{P}}(e_\eta, e_{\eta 0}, e_{\eta 1} \,|\, lc_\eta, lc_{\eta 0}, lc_{\eta 1}) \cdot \tilde{\mathsf{P}}(lc_\eta, lc_{\eta 0}, lc_{\eta 1}) \tag{4.7}$$

M-step:

$$\hat{\mathsf{P}}_{\theta_{\text{M}}}(lce_\eta \to lc_{\eta 0}, lc_{\eta 1}) \leftarrow \frac{\sum_{e_{\eta 0}, e_{\eta 1}} \tilde{\mathsf{P}}(lce_\eta, lce_{\eta 0}, lce_{\eta 1})}{\sum_{cle_{\eta 0}, cle_{\eta 1}} \tilde{\mathsf{P}}(lce_\eta, lce_{\eta 0}, lce_{\eta 1})} \tag{4.8}$$

$$\hat{\mathsf{P}}_{\theta_{\text{L}}}(e_{\eta 0} \,|\, e_\eta; l_{\eta 0}) \leftarrow \frac{\sum_{cl_\eta, c_{\eta 0}, cle_{\eta 1}} \tilde{\mathsf{P}}(lce_\eta, lce_{\eta 0}, lce_{\eta 1})}{\sum_{cl_\eta, ce_{\eta 0}, cle_{\eta 1}} \tilde{\mathsf{P}}(lce_\eta, lce_{\eta 0}, lce_{\eta 1})}, \quad \text{also for } e_{\eta 1} \tag{4.9}$$

$$\hat{\mathsf{P}}_{\theta_{\text{H}}}(h_\eta \,|\, lce_\eta) \leftarrow \frac{\tilde{\mathsf{P}}(lce_\eta, -, -)}{\sum_{h_\eta} \tilde{\mathsf{P}}(lce_\eta, -, -)}, \quad \text{when } \eta \text{ is a preterminal} \tag{4.10}$$

The first line of the E-step uses inside and outside probabilities to estimate the behavior of abstract concepts at any binary branch in the tree (preterminals are simliar, but with null constituents for the children). Inside probabilities can be recursively calculated on training trees from the bottom up. These are probability sums of all subsumed subtrees; they are in fact the same thing as Viterbi probabilities with sums instead of maxes. To indicate a whole subsumed span rather than just a lone constituent, they are written with a bar.

$$\begin{aligned}\hat{\mathsf{P}}_{\theta_{\text{Ins}}}(\overline{lch_\iota} \,|\, lce_\iota) = \hat{\mathsf{P}}_{\theta_{\text{M}}}(lce_\iota \to lc_{\iota 0}, lc_{\iota 1}) \sum_{e_{\iota 0}} \hat{\mathsf{P}}_{\theta_{\text{L}}}(e_{\iota 0} \,|\, e_\iota; l_{\iota 0},) \cdot \hat{\mathsf{P}}_{\theta_{\text{Ins}}}(\overline{lch_{\iota 0}} \,|\, lce_{\iota 0}) \\ \cdot \sum_{e_{\iota 1}} \hat{\mathsf{P}}_{\theta_{\text{L}}}(e_{\iota 0} \,|\, e_\iota; l_{\iota 0}) \cdot \hat{\mathsf{P}}_{\theta_{\text{Ins}}}(\overline{lch_{\iota 1}} \,|\, lce_{\iota 1})\end{aligned} \tag{4.11}$$

Inside probabilities can also be written $\hat{\mathsf{P}}_{\theta_{\text{Ins}}}(h_\iota \,|\, lce_\iota)$ since lc are deterministically copied from the condition and h_ι always subsumes its progeny, but writing the complete target makes it easier to see how inside probabilities fit into the E-step.

Outside probabilities can also be recursively calculated from training trees, here

from parent probabilities. For a left child (the right-child case is similar):

$$\hat{\mathsf{P}}_{\theta_{\text{Out}}}(lce_{\iota 0}, \overline{lch_{\epsilon}}\text{–}\overline{lch_{\iota 0}}) = \hat{\mathsf{P}}_{\theta_{\text{Out}}}(lce_{\iota}, \overline{lch_{\epsilon}}\text{–}\overline{lch_{\iota}}) \cdot \hat{\mathsf{P}}_{\theta_{\text{M}}}(lce_{\iota} \to lc_{\iota 0}, lc_{\iota 1}) \cdot \sum_{e_{\iota 1}} \hat{\mathsf{P}}_{\theta_{\text{L}}}(e_{\iota 1} \,|\, e_{\iota}; l_{\iota 1}) \cdot \hat{\mathsf{P}}_{\theta_{\text{Ins}}}(\overline{lch_{\iota 1}} \,|\, lce_{\iota 1}) \cdot \hat{\mathsf{P}}_{\theta_{\text{L}}}(e_{\iota 0} \,|\, e_{\iota}; l_{\iota 0}) \quad (4.12)$$

Since outside probabilities signify everything *but* what is subsumed by the node, they carry a complementary set of information to inside probabilities. Thus, inside and outside probabilities together are a natural way to produce parent and child abstract concepts (Equation 4.6). The probability of this concept-generating process is then weighed together with the probability of seeing the configuration of variables in the empirical data (Equation 4.7).

In the M-step, the data-informed result of the E-step is used to update the estimates of θ_{M}, θ_{L}, and θ_{H}. These updated estimates are then plugged back in to the next E-step. The two steps continually alternate until some convergence condition is reached, or until a maximum number of iterations.

4.3 Relational Semantic Clusters in Parsing

Section 4.1 started this chapter by listing the five probability models that constitute an interface for instantiations of SVS. Because abstract concepts e cannot be empirically determined, EM was introduced as a way to estimate θ_{M}, θ_{L}, $\theta_{\text{P-Vit(G)}}$, $\pi_{\text{G}\epsilon}$, and π_{G}.

The estimate in Equation 4.8 can directly be used for θ_{M}, and Equation 4.9 can be used for θ_{L}. The other three models also fall out nicely from the algorithm, though they are not explicitly estimated in EM.

First, the preterminal model is based on θ_{H}, but it also includes some backoff for the case of words that have not been used as headwords.

$$\hat{\mathsf{P}}_{\theta_{\text{P-Vit(G)}}}(x_{\eta} \,|\, lce_{\eta}) = \begin{cases} \hat{\mathsf{P}}_{\theta_{\text{H}}}(x_{\eta} \,|\, lce_{\eta}) & x_{\eta} \in H \\ \mathsf{P}_{\theta_{\text{P-Vit(G)}}}(x_{\eta} \,|\, c_{\eta}) \cdot \hat{\mathsf{P}}_{\theta_{\text{H}}}(\texttt{unk} \,|\, lce_{\eta}) & x_{\eta} \notin H \end{cases} \quad (4.13)$$

Next, the prior probability at the root is in fact the base case for outside probabilities:

$$\hat{\mathsf{P}}_{\pi_{\text{G}\epsilon}}(lce_{\epsilon}) \stackrel{\text{def}}{=} \hat{\mathsf{P}}_{\theta_{\text{Out}}}(lce_{\epsilon}, \overline{lch_{\epsilon}}\text{–}\overline{lch_{\epsilon}}) \quad (4.14)$$

Finally, prior probabilities at other constituents are easily estimated from the empirically-weighted joint probability in Equation 4.7. In fact, this probability is used in the denominator of θ_{M} in the M-step (Equation 4.8).

$$\hat{\mathsf{P}}_{\pi_{\mathrm{G}}}(lce_{\eta}) \stackrel{\text{def}}{=} \sum_{lce_{\eta 0}, lce_{\eta 1}} \tilde{\mathsf{P}}(lce_{\eta}, lce_{\eta 0}, lce_{\eta 1}) \tag{4.15}$$

With a full definition of these models, a relationally-clustered SVS parser is now defined.

4.4 Evaluation

Features of the relational-clustering SVS implementation are discussed first, followed by a series of experiments teasing apart different parameters in the model.

4.4.1 Implementation

As in Chapter 3, Sections 02–21 of the Wall Street Journal (WSJ) corpus were used as training data; Section 23 was used as test data. Punctuation was left in for all reported evaluations. Trees were binarized; then, each branch was annotated with a head relation $l_{\textsc{Id}}$ or a modifier relation $l_{\textsc{Mod}}$ according to a binarized version of headword percolation rules (Magerman, 1995; Collins, 1997), and the headword was propagated up from its head constituent. The most frequent headwords (e.g., $h_1, \ldots, h_{50}$) were stored, and the rest were assigned a constant, 'unknown' headword category.

From counts on the binary rules of these annotated trees, the θ_{M}, θ_{L}, $\theta_{\text{P-Vit(G)}}$, $\pi_{\mathrm{G}\epsilon}$, and π_{G} probabilities for headword lexicalization (as in Section 3.4) were obtained. Modifier relations $l_{\textsc{Mod}}$ were deterministically augmented with their syntactic context; both c and l symbols appearing fewer than 10 times in the whole corpus were assigned 'unknown' categories.

These lexicalized models served as a baseline, but the augmented trees from which they were derived were also inputs to the EM algorithm in Section 4.2. Each parameter in the model or training algorithm was examined, with $|E| = \{1, 5, 10, 15, 20\}$ clusters, random initialization from reproducible seeds, and a varying numbers of EM iterations.

Depending on the number of headwords retained and the random initial seed, the resulting models varied a great deal in size (and therefore, training time). On average,

Cluster e_0 'money'		**Cluster** e_1 'people'		**Cluster** e_2 'companies'		**Cluster** e_5 'time'	
unk	0.431	officials	0.145	unk	0.248	years	0.244
cents	0.135	unk	0.141	markets	0.056	months	0.187
shares	0.084	years	0.132	companies	0.036	unk	0.183
yen	0.036	shares	0.093	issues	0.035	days	0.115
sales	0.025	prices	0.061	firms	0.033	weeks	0.056
points	0.023	people	0.050	banks	0.030	points	0.028
marks	0.018	stocks	0.032	loans	0.025	companies	0.020
francs	0.018	sales	0.027	investors	0.024	hours	0.015
tons	0.013	executives	0.024	contracts	0.022	people	0.013
people	0.012	analysts	0.018	stocks	0.021	units	0.011

Cluster e_1 'announcement'		**Cluster** e_5 'change in value'		**Cluster** e_7 'change possession'	
unk	0.362	rose	0.137	unk	0.381
was	0.173	fell	0.124	had	0.065
reported	0.097	unk	0.116	was	0.062
posted	0.036	gained	0.063	took	0.036
earned	0.029	dropped	0.051	bought	0.027
filed	0.024	attributed	0.051	completed	0.025
were	0.022	jumped	0.046	received	0.024
had	0.020	added	0.041	were	0.023
told	0.013	lost	0.039	got	0.018
approved	0.013	advanced	0.022	made	0.018
				acquired	0.016

Table 4.1: Example θ_{H} clusters from 1,000 headwords clustered into 10 referents, after 10 EM iterations. First row: plural nouns, NNS. Second row: transitive past-tense verbs, VBD-argNP.

clustered models are much larger than syntax-only or headword-lexicalized models because semantic information can be shared between concepts, smoothing over missing headwords or unseen similar headwords. However, there is some variability because EM learns some clusters to be highly distributed, whereas other clusters have very specific roles; learning many good, specific clusters will lead to a smaller model.

The implemented parser had few adjustments from a plain CKY parser other than these vectors. No approximate inference was used, with no beam for candidate parses and no re-ranking.

4.4.2 Interpretable and cohesive relational clusters

Table 4.1 shows example clusters for one of the headword models used, where EM clustered 1,000 headwords into 10 concepts in 10 iterations. The lists shown are parts of the $\hat{\mathsf{P}}_{\theta_{\mathrm{H}}}(h_\eta \,|\, lce_\eta)$ model. As such, each of the 10 clusters will only produce headwords in light of some syntactic constituent. The first row in the table shows how abstract concepts produce headwords for plural nouns, and the second shows transitive past-tense verbs. Note that the probability distributions for different headwords are quite uneven, again confirming that some clusters are more specific, and others are more general.

Each cluster has been given a heading of its approximate meaning — e_5, for example, includes many words that measure 'time' when in the context of a plural noun. Interestingly, in the context of a past-tense verb, e_5 mostly picks verbs that are 'change in value' events that would frequently be time-sensitive in a corpus like the Wall Street Journal. Although any other concept may be a modifier of e_5, the general meaning of that particular cluster could be seen to include both ideas. This points at the cohesiveness of the semantic component in this model, though this cohesiveness is formally embodied by the sharing of concepts across syntactic contexts.

4.4.3 Dependency on Initial Values

Since EM is seeded randomly, four different random initializations of the same model are compared on parsing performance to determine whether the initialization matters. Using 500 headwords clustered into 5 concepts, Table 4.2 gives the labeled precision (LP), labeled recall (LR), and F-score (F) of the 5 randomly-initialized models on WSJ Section 23, along with their averages and standard deviations.

	1	2	3	4	$\mu \pm \sigma$
LR	83.60	84.15	83.83	83.92	83.88±0.23
LP	83.84	84.31	84.12	84.08	84.09±0.19
F	83.72	84.23	83.98	84.00	83.98±0.21

Table 4.2: Performance of four different random initializations on WSJ 23 for 500-headword, 10-cluster models.

The results show that random initialization may have some effect on individual models in head-to-head comparison. It is important to remember that a model does

Sec. 23, < 40	LR	LP	F
baseline 00hw:	83.32	83.83	83.57
headword-lex. 10hw:	83.10	83.61	**83.35**
headword-lex. 20hw:	83.15	83.65	83.40
headword-lex. 50hw:	83.09	83.40	**83.24**
Relational-clustering			
50hw → 10e:	83.67	84.13	**83.90**
100hw → 10e:	83.57	83.80	83.68
500hw → 10e:	83.60	83.84	83.72
1khw → 10e:	83.75	84.04	83.89
5khw → 10e:	82.91	83.05	82.98

Table 4.3: Recall, precision (LP), and weighted average (F-score) results for unsmoothed lexicalized CKY parsers (top) versus parsers with 10 semantic clusters (bottom). Evaluations were run on WSJ Section 23 with punctuation, for sentences less than 40 words, with EM trained to 10 iterations.

not change after it is trained; thus, in practical use, the best-performing model of these could always be chosen. Alternatively, it may be possible to pick a better starting point for EM by using heuristics.

4.4.4 Comparison to Lexicalization

One important comparison to draw here is between the effectiveness of semantic clusters versus headword-lexicalization. For fair head-to-head comparison on WSJ Section 23, both models were vectorized and included no smoothing or backoff. Neither relational clusters nor lexicalization were optimized with manually-added features beyond binarization.

Table 4.3 shows precision, recall, and F-score for lexicalized models and for clustered semantic models. Consider the 50-headword, 10-cluster model (in bold). It would be natural to compare this model to the headword-lexicalized model with 50 headwords, since the same information from the trees is available to both models. Clearly, the relationally-clustered model outperforms the headword-lexicalized model, showing that clustering the headwords actually improves their usefulness, despite the fact that fewer referents are used in the actual vectors.

It is also interesting, then, to compare this 50-headword, 10-cluster model to a

headword-lexicalized model with 10 headwords. In this case, the possible size of the grammar is equal. Again, the relationally-clustered model outperforms plain lexicalization. This indicates that the 10 clustered referents are much more meaningful than 10 headword referents for the disambiguating of syntax.

Compared to the 00-headword case (an unlexicalized parser), clustering from almost any number of headwords will improve parsing accuracy. The one outlier is the 5,000 headword model, which likely suffers from sparse data effects. As a whole, there is no clear trend between number of headwords and parsing accuracy.

4.4.5 Effect of Vectorization on Speed

A CKY parser, as used for this implementation, has worst-case $\mathcal{O}(n^3 \cdot |L|^3|C|^3|E|^3)$ runtime. While this may be acceptable in typical lexicalized models, the factorization from Section 3.1 additionally requires up to $2|E|^2+3|E|$ multiplications for each syntactic rule. With dense relational distributions, clusters incur a larger cost than the comparatively sparse information found in bilexical dependencies.

Of course, both models may be sped up considerably by using coarse-to-fine parsing, pruning, and other methods. But for vectorial models, contiguous storage of information intuitively leads to more efficient multiplication of relevant probabilities. Thus, improvements in parsing speed for dense, relationally-clustered semantics may be an implementational windfall from vectorization.

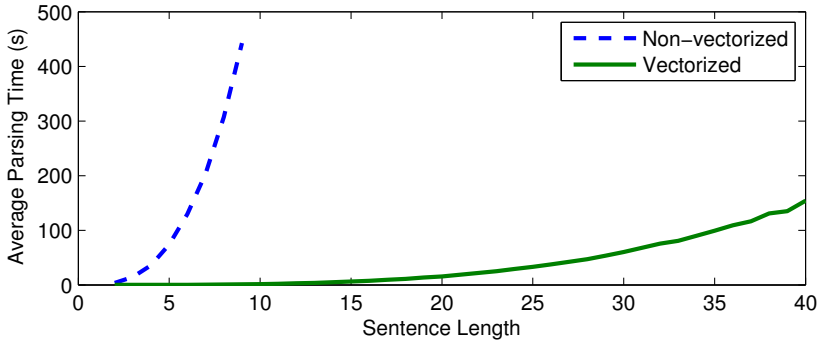


Figure 4.2: Speed of relationally-clustered SVS parsers, with and without vectorization.

The plot in Figure 4.2 compares simple CKY implementations of a relationally-clustered SVS model with and without vectorization on sentences (ordered by length) on

WSJ Section 23 without punctuation. The un-vectorized, relationally-clustered model was cut off because it was impractically slow on longer sentences. Regression analysis shows that these are indeed cubic; the constant in front of $\mathcal{O}(n^3)$ is 0.66505 without vectorization, and 0.00267 with vectorization. In other words, the cost of accessing the grammatical constituents and performing the factored multiplications was greatly reduced by vectorization.

4.4.6 Effect of EM Iterations

Tuning relationally-clustered SVS requires a principled way to find the best number of EM iterations. Table 4.4 shows how the same model evolves over the course of the EM algorithm, with respect to parsing accuracy on WSJ Section 22 (the development set). It should be kept in mind that each line in the table uses the same initial random seed, and that the algorithm proceeds by always increasing the log-likelihood of the data.

Sec. 22, < 40	LR	LP	F
2 iterations	85.59	85.06	85.32
4 iterations	85.57	85.31	85.44
6 iterations	85.17	85.00	85.08
8 iterations	84.29	84.16	84.22
10 iterations	84.25	84.07	84.16

Table 4.4: Dependence of relational clustering on EM iterations for WSJ Section 22 with punctuation, on sentences less than 40 words.

Despite the monotonic increase in log-likelihood over iterations, parsing accuracy does not increase similarly. The model can easily overfit the data. The table shows that 4 iterations give a peak of performance, and that further iterations fall prey to overfitting.

4.4.7 Effect of Number of clusters

The final experiment on relational-clustering SVS was to determine whether performance would vary with the number of clusters. Figure 4.3 compares average performance (over different random initializations) for numbers of clusters from 1 (a syntax-equivalent case) to 20.

Sec. 23, < 40	LR	LP	F
baseline, 1e	83.34	83.90	83.62
1khw, 5e, avg	83.85	84.23	84.04
1khw, 10e, avg	84.04	84.40	84.21
1khw, 15e, avg	84.15	84.38	84.26
1khw, 20e, avg	84.21	84.42	**84.31**
1khw, 5e, max	84.40	84.57	84.49
1khw, 10e, max	84.15	84.62	84.38
1khw, 15e, max	84.33	84.74	84.53
1khw, 20e, max	84.48	84.72	**84.60**

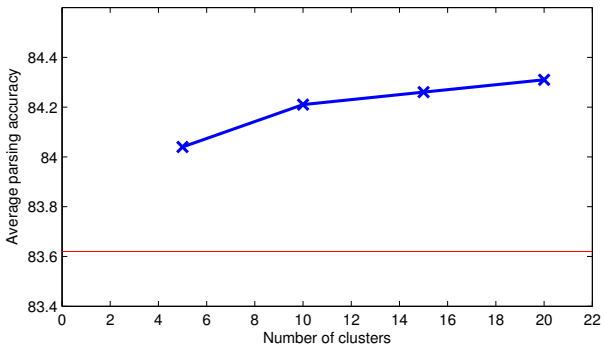


Figure 4.3: Dependence of relational clustering on number of semantic clusters, for WSJ Section 23 with punctuation on sentences less than 40 words. Average values are taken over different random seeds, with EM running 4 or 10 iterations. Maximum values show the best performance by a single model in each class.

First, it should be noted that all of the relationally clustered models improved on the baseline. Random initializations did not vary enough for these models to do worse than syntax alone. For each vector/domain size, in fact, the gains over syntax-only are substantial.

In addition, the plot shows that average performance increases with the number of clusters. This loosely positive slope means that EM is still finding useful parts of the semantic space to explore and cluster, so that the clusters remain meaningful. However, it does seem as though the increase in performance with number of clusters will eventually plateau.

Maximum-accuracy models are reported in Figure 4.3 as well, since any EM-trained is a full-fledged parser. The best 20-referent model beats the syntactic baseline by almost a full absolute point. Thus, finding relationally-clustered semantic output also contributes to some significant parsing benefit.

4.5 Summary and Discussion

This chapter has presented a method for clustering sparse, overly-specific lexicalized data into semantic concepts. The key contribution of this approach is that a quantitative, distributed semantic representation has been fully interleaved with syntactic

processing by defining vector composition in syntactic context.

Evaluations on a Structured Vectorial Semantic parser with these concepts show that including a rich-dependency clustered model outperforms headword-only lexicalized models of similar size. A mildly positive effect of larger number of clusters was also observed.

In lieu of optimizing the number of clusters for EM dimensionality reduction, future work may replace relational clustering with any number of other topic modeling type techniques. For example, a non-parametric Bayesian model may be applied to the relational data, allowing for an automatic and appropriate choice for number of clusters.

Alternatively, the model may be used for related tasks like semantic role labeling or coreference resolution. The former can directly use the relations l to express semantic role as a generalization of the head–modifier relationship. This would result in full parses that additionally defined semantic role. Coreference resolution could also be done by using both the semantic and syntactic output in this model; if it is assumed that mentions are given, resulting distributions over concepts at each mention's subtree root could be composed with other mentions' concept mixture.

Chapter 5

Logical Interpretation with SVS

In motivating Structured Vectorial Semantics, the strategy of extracting latent information from lexical items has been liberally used. This does a good job of getting the 'gist' meaning of utterances, but ignores prior knowledge (other than syntactic Treebank data) and excludes the possibility of any formal reasoning over precise referents (such as named entities).

The aim of this chapter, then, is to demonstrate how SVS can encode logical, model-theoretic interpretation. In model theory (Tarski, 1933; Church, 1940), a *world model* is defined as a tuple containing: (1) a domain of individuals, and (2) an interpretation function to interpret expressions in terms of those individuals. Expressions will consist of unary or binary logical predicates, and the interpretation will ultimately return truth values; or, as in stochastic lambda calculus (Ramsey and Pfeffer, 2002), it may return probabilities of the predicates' truth values. All predicates in logical-interpretation SVS will be grounded in this type of world model.

Probability distributions were encapsulated in vectors and matrices in Chapter 3; here, those vectors and matrices will define set membership. By defining special relations and structure for some basic operations of first-order predicate logic, it will be evident that SVS is general enough to subsume this kind of semantic representation. In order to accomplish this, the vectorized representation of SVS will be taken to be canonical, rather than the probability representation. Then, non-probabilities may be used in the vectors and matrices in order to embody the necessary logical operators.

Because of the difference in approach for representing meaning, this chapter will use

referent or *entity* to describe the semantic component, rather than the more abstract 'concept.' The same SVS framework will be used to describe these different meaning representations; in future work, the difference between the two may be blurred such that entities bear the properties of abstract concepts.

This model-theoretic semantic framework, laid alongside syntactic parsing, is similar to previous work on some spoken language interfaces (Wu, Schwartz, and Schuler, 2008b; Schuler, Wu, and Schwartz, 2009). These systems begin with defined world models, and a user's speech is used to reason over the proper individuals in the world models. The parsers used in these models operate incrementally, as will be described in Chapter 7. With an incremental version of logical interpreation in SVS, such speech interfaces could actually be implemented with SVS.

The task of learning the meaning representation (MR) is an important one, but it is beyond the scope of this chapter. Solid models for semantic parsing, which extracts formal meaning representations[1] from text, have been successfully learned (Ge and Mooney, 2005; Mooney, 2007; Kate, 2008; Ge and Mooney, 2009). These models may be integrated with the SVS logical interpretation framework, which would then simultaneously consider syntax and learned formal semantics simultaneously. Currently, however, interpretation with semantic parsers are pipelined systems.

The rest of the chapter will begin by casting vectors and matrices as sets (Section 5.1), then present linear-algebraic versions of logical operations (Section 5.2), interface the logical constructs with SVS (Section 5.3), and present an extended, illustrative example (Section 5.4).

5.1 Vectors and Matrices as Sets

In a given world model, we will assume a domain E of individual entities. Vector indices, then, will exhaustively list these entities. With indices tied to individuals in the domain, vectors define sets of denoted individuals (i.e. functors from individuals to set-membership truth values).

[1] Semantic role labeling (Gildea and Jurafsky, 2002) and other related tasks may be considered forms of 'shallow' semantic parsing.

Definition (Vector Sets)**.** *For a vector* $\mathbf{e}$*, its indices* $e \in E$*, and an associated set* $S \subseteq E$*:*

$$\mathbf{e} \stackrel{\text{def}}{=} e \mapsto e \in S \tag{5.1}$$

and the predicate $S(e) = e \in S$ *returns the set-membership truth value (or probability).*

Similarly, matrices define relations over individuals (i.e. functors from individuals to individuals), encoding sets of ordered pairs.

Definition (Matrix Sets)**.** *For a matrix* $\mathbf{L}$*, its row and column indices* $e, e' \in E$*, and an associated set of ordered pairs* $R \subseteq E \times E$*:*

$$\mathbf{L} \stackrel{\text{def}}{=} e \mapsto e' \mapsto \langle e, e' \rangle \in R \tag{5.2}$$

and the predicate $R(e, e') = \langle e, e' \rangle \in R$ *returns the set-membership truth value (or probability).*

With $E = \{\mathrm{e}_1, \mathrm{e}_2, \mathrm{e}_3, \mathrm{e}_4\}$, an example vector and matrix might be:

$$\begin{array}{c} \\ \textit{individuals} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_4 \end{array}\!\!\begin{array}{c} \textit{truth} \\ \left[\begin{array}{c} 1 \\ 0 \\ 1 \\ 1 \end{array}\right] \end{array} = \{\mathrm{e}_1, \mathrm{e}_3, \mathrm{e}_4\} \qquad \begin{array}{c} \\ \textit{domain} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_4 \end{array}\!\!\begin{array}{c} \textit{codomain} \\ \begin{array}{cccc} \mathrm{e}_1 & \mathrm{e}_2 & \mathrm{e}_3 & \mathrm{e}_4 \end{array} \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \end{array}\right] \end{array} = \{\langle \mathrm{e}_2, \mathrm{e}_3 \rangle, \langle \mathrm{e}_3, \mathrm{e}_1 \rangle, \langle \mathrm{e}_3, \mathrm{e}_3 \rangle, \langle \mathrm{e}_4, \mathrm{e}_2 \rangle\} \tag{5.3}$$

For $e = \mathrm{e}_3$ in the matrix above, and wherever there are multiple elements in the codomain for each element of the domain, probabilities may be assigned uniformly.

Matrix multiplication defines a composition of these functors:

$$\begin{array}{c} \\ \textit{domain } \alpha \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_4 \end{array}\!\!\begin{array}{c} \textit{codomain } \beta \\ \begin{array}{cccc} \mathrm{e}_1 & \mathrm{e}_2 & \mathrm{e}_3 & \mathrm{e}_4 \end{array} \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \end{array}\right] \end{array} \cdot \begin{array}{c} \textit{codomain } \gamma \\ \textit{(truth)} \\ \\ \textit{domain } \beta \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_4 \end{array}\!\!\begin{array}{c} \\ \left[\begin{array}{c} 1 \\ 0 \\ 1 \\ 1 \end{array}\right] \end{array} = \begin{array}{c} \textit{codomain } \gamma \\ \textit{(truth)} \\ \\ \textit{domain } \alpha \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_4 \end{array}\!\!\begin{array}{c} \\ \left[\begin{array}{c} 0 \\ 1 \\ 1 \\ 0 \end{array}\right] \end{array} = \{\mathrm{e}_2, \mathrm{e}_3\} \tag{5.4}$$

$$(\alpha \to \beta) \quad \circ \quad (\beta \to \gamma) \;=\; (\alpha \to \gamma)$$

picking out the individuals from the domain (α) of the first functor that are related through that functor to individuals in the domain (β) of the second. Thus, individuals e from $R(e, e')$ are retained in the set composition only if e' is found in $S(e')$; so, e_4 is dropped in Equation 5.4.

5.2 Logical operators

In this framework, we can now define the typical operations in first-order predicate logic: conjunction, negation, and universal quantification. From these, all other statements can be derived.

Although definitions have involved sets, the quantities defined are actually truth values (or membership probabilities) over sets. Thus, we will use both set terminology and logical terminology, making the connections wherever they are not obvious.

Definition (Conjunction)**.** *For two vectors* $\mathbf{e}_\eta$ *and* $\mathbf{e}_\iota$*, the conjunction is the dot product*

$$\mathbf{e}_\eta \wedge \mathbf{e}_\iota = \mathrm{d}(\mathbf{e}_\eta) \cdot \mathrm{d}(\mathbf{e}_\iota) \cdot \mathbf{1} \tag{5.5}$$

This is the same as set intersection, where membership (a non-zero value) in both $\mathbf{e}_\eta$ and $\mathbf{e}_\iota$ is requisite for membership in the resulting set. In Structured Vectorial Semantics, two vectors $\mathbf{e}_{\eta 0} = [0\ 1\ 1\ 1]^T$ and $\mathbf{e}_{\eta 1} = [1\ 1\ 0\ 1]^T$ could first be multiplied by identity relations $\mathbf{L}_{\mathrm{ID}} = \mathbf{I}$ before being conjoined:

$$\begin{array}{c} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_4 \\ \\ \mathrm{d}(\mathbf{L}_{\mathrm{ID}} \cdot \mathbf{e}_{\eta 0}) \end{array} \begin{array}{c} \begin{array}{cccc} \mathrm{e}_1 & \mathrm{e}_2 & \mathrm{e}_3 & \mathrm{e}_4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \cdot \begin{array}{c} \begin{array}{cccc} \mathrm{e}_1 & \mathrm{e}_2 & \mathrm{e}_3 & \mathrm{e}_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \cdot \begin{array}{c} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_4 \\ \\ \mathbf{1} \end{array} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{array}{c} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_4 \\ \\ \mathbf{e}_\eta \end{array} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \tag{5.6}$$

Negation (set complementation) can be performed by adding a 'shadow' element for every entity. We will continue to refer to vectors without changing notation, but vector indices of shadow elements will be denoted as e' and appended at the bottom of matrices and vectors.

Definition (Negation)**.** *Let* $\mathbf{e}_\eta$ *be a vector with shadow elements repeating the original vector. Let* $\mathbf{L}_{\mathrm{NEG}}$ *be a matrix with block structure*

$$\mathbf{L}_{\mathrm{NEG}} = \left[\begin{array}{c|c} -\mathbf{I} & \mathbf{I} \\ \hline \bar{\bar{\mathbf{0}}} & \mathbf{I} \end{array}\right],$$

where each block dimension is the size of $\mathbf{e}_\eta$ *without shadow elements. The negation of* $\mathbf{e}_\eta$ *is*

$$\neg \mathbf{e}_\eta = \mathbf{L}_{\mathrm{NEG}} \cdot \mathbf{e}_\eta \tag{5.7}$$

As an example of negation:

$$\begin{array}{c} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}'_1 \\ \mathrm{e}'_2 \\ \mathrm{e}'_3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc|ccc} \mathrm{e}_1 & \mathrm{e}_2 & \mathrm{e}_3 & \mathrm{e}'_1 & \mathrm{e}'_2 & \mathrm{e}'_3 \end{array} \\ \left[\begin{array}{ccc|ccc} -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \\ \mathbf{L}_{\mathrm{NEG}} \end{array} \cdot \begin{array}{c} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}'_1 \\ \mathrm{e}'_2 \\ \mathrm{e}'_3 \end{array}\!\!\begin{array}{c} \textit{truth} \\ \left[\begin{array}{c} 0 \\ 0 \\ 1 \\ \hline 1 \\ 1 \\ 1 \end{array}\right] \\ \mathbf{e}_\eta \end{array} = \begin{array}{c} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}'_1 \\ \mathrm{e}'_2 \\ \mathrm{e}'_3 \end{array}\!\!\begin{array}{c} \textit{truth} \\ \left[\begin{array}{c} 1 \\ 1 \\ 0 \\ \hline 1 \\ 1 \\ 1 \end{array}\right] \\ \neg\mathbf{e}_\eta \end{array} \tag{5.8}$$

With set definitions and the conjunction operation, the contents of every vector and matrix could be considered a valid probability distribution or likelihood function. But $\mathbf{L}_{\mathrm{NEG}}$ clearly requires non-probabilities, showing that semantic relations in logical interpretation require a formalism like linear algebra that is more flexible than pure probabilities.

To reason in first-order logic rather than just propositional logic, we must further break from using vectors and matrices as probability distributions by defining an additional (possibly non-linear) operator $\mathbf{n}$ that functions in place of $\mathbf{1}$. For existential quantifiers, $\mathbf{n}=\mathbf{1}$. But for non-existential quantifiers, we will need a cardinality test to determine whether the correct number of referents is denoted. To that end, we define a thresholding operator $\mathbf{n}_{\mathrm{TH}}$ to act on a matrix $\mathbf{E}$, comparing row sums of regular elements to row sums of shadow elements.[2]

$$\mathbf{E}\cdot\mathbf{n}_{\mathrm{TH}} = \mathbf{e} \quad \text{s.t. } \mathbf{e}[\mathrm{e}_i] = \begin{cases} \sum_{\mathrm{e}_j} \mathbf{E}[\mathrm{e}_i,\mathrm{e}_j], & \text{if } \sum_{\mathrm{e}_j} \mathbf{E}[\mathrm{e}_i,\mathrm{e}_j] \geq \sum_{\mathrm{e}''_j} \mathbf{E}[\mathrm{e}_i,\mathrm{e}''_j] > 0 \\ 0, & \text{otherwise} \end{cases} \tag{5.9}$$

Then, we can define universal quantification, keeping in mind that everything is related to the root referent by successive binary relations.

Definition (Universal Quantification within a binary relation)**.** *Let S be a unary predicate represented by vector $\mathbf{e}_\iota$ with shadow elements equal to 1. Let R be a binary predicate with associated relation matrix $\mathbf{L}_R$, with shadow elements in block-diagonal structure*

$$\mathbf{L}_{\mathrm{R}} = \left[\begin{array}{c|c} \mathbf{L}^+_{\mathrm{R}} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{L}'_{\mathrm{R}} \end{array}\right]$$

where $\mathbf{L}^+_{\mathrm{R}}$ is a relation matrix according to Equation 5.2, and $\mathbf{L}'_R$ is a matrix with average values of each row in $\mathbf{L}^+_R$ stored on the diagonal. Let $\mathbf{L}_{\mathrm{ALL}}$ be a matrix with

[2] Different shadow elements are used for negation and quantification, so the complete representations are actually 3×3 block matrices. The extra blocks are omitted for clarity.

block structure

$$\mathbf{L}_{\text{ALL}} = \left[\begin{array}{c|c} \mathbf{I} & \mathbf{0} \\ \hline \mathbb{1} & \mathbf{0} \end{array}\right]$$

where $\mathbb{1}$ *is a matrix of all ones. Then universal quantification over the binary relation is:*

$$\mathbf{e}_\eta \;=\; e_\eta \mapsto \big(\; \forall e_\iota S(e_\iota) \Rightarrow R(e_\eta, e_\iota) \;\big) \;=\; \mathrm{d}(\mathbf{L}_R \cdot (\mathbf{L}_{\text{ALL}} \cdot \mathbf{e}_\iota)) \cdot \mathbf{n}_{\text{TH}} \tag{5.10}$$

We give an abbreviated example of a universal quantifier:

$$\mathrm{d}\left(\begin{array}{c} \begin{array}{c} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_1'' \\ \mathrm{e}_2'' \\ \mathrm{e}_3'' \end{array} \overset{\begin{array}{cccccc} \mathrm{e}_1 & \mathrm{e}_2 & \mathrm{e}_3 & \mathrm{e}_1'' & \mathrm{e}_2'' & \mathrm{e}_3'' \end{array}}{\left[\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]} \\ \mathbf{L}_R \end{array} \cdot \left(\begin{array}{c} \begin{array}{c} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_1'' \\ \mathrm{e}_2'' \\ \mathrm{e}_3'' \end{array} \overset{\begin{array}{cccccc} \mathrm{e}_1 & \mathrm{e}_2 & \mathrm{e}_3 & \mathrm{e}_1'' & \mathrm{e}_2'' & \mathrm{e}_3'' \end{array}}{\left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{array}\right]} \\ \mathbf{L}_{\text{ALL}} \end{array} \cdot \begin{array}{c} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_1'' \\ \mathrm{e}_2'' \\ \mathrm{e}_3'' \end{array} \overset{\begin{array}{c} \textit{truth} \end{array}}{\left[\begin{array}{c} 1 \\ 0 \\ 1 \\ \hline 1 \\ 1 \\ 1 \end{array}\right]} \\ \mathbf{e}_S \end{array}\right)\right) \cdot \mathbf{n}_{\text{TH}}$$

$$= \begin{array}{c} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_1'' \\ \mathrm{e}_2'' \\ \mathrm{e}_3'' \end{array} \overset{\begin{array}{cccccc} \mathrm{e}_1 & \mathrm{e}_2 & \mathrm{e}_3 & \mathrm{e}_1'' & \mathrm{e}_2'' & \mathrm{e}_3'' \end{array}}{\left[\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \frac{2}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{array}\right]} \cdot \mathbf{n}_{\text{TH}} \;=\; \begin{array}{c} \\ \mathrm{e}_1 \\ \mathrm{e}_2 \\ \mathrm{e}_3 \\ \mathrm{e}_1'' \\ \mathrm{e}_2'' \\ \mathrm{e}_3'' \end{array} \overset{\begin{array}{c} \textit{truth} \end{array}}{\left[\begin{array}{c} 0 \\ 1 \\ 0 \\ \hline 1 \\ 1 \\ 1 \end{array}\right]} \tag{5.11}$$

$$\mathbf{e}_{R\cdot\,\text{ALL}\cdot\,\text{S}}$$

where $\mathbf{L}'_R$ has value $1/|E|$ when the corresponding row in $\mathbf{L}^+_R$ has all zeros. The operations have selected e_2 as the only individual that is related (via R) to both e_1 and e_3 (which are all the entities in S).

With these operations, this model is able to perform model-theoretic interpretation in first-order logic. Bringing these together with syntax in the context of natural language will be demonstrated via extended example in Section 5.4.

5.3 SVS Language Model for Logical Interpretation

Structured Vectorial Semantic language model may be defined using the foregoing sets and logical operators. As before, vectors and matrices are initially populated by language model probabilities.

$$\mathbf{M}(lc_\eta \to lc_{\eta 0}\, lc_{\eta 1}) = e_\eta \mapsto e'_\eta \mapsto \mathsf{P}_{\theta_{\mathrm{M}}}(lce_\eta \to lc_{\eta 0}\, lc_{\eta 1})[\![e_\eta = e'_\eta]\!] \tag{5.12}$$

$$\mathbf{L}^+_\iota = e_\eta \mapsto e_\iota \mapsto \mathsf{P}_{\theta_{\mathrm{L}}}(e_\iota \,|\, e_\eta; l_\iota) \tag{5.13}$$

$$\mathbf{e}_\eta = e_\eta \mapsto \mathsf{P}_{\theta_{\text{P-Vit(G)}}}(x_\eta \,|\, lce_\eta) \tag{5.14}$$

$$\mathbf{a}^T_\epsilon = e_\epsilon \mapsto \mathsf{P}_{\pi_{\mathrm{G}\epsilon}}(lce_\epsilon) \tag{5.15}$$

$$\mathbf{a}^T_\eta = e_\eta \mapsto \mathsf{P}_{\pi_{\mathrm{G}}}(lce_\eta) \tag{5.16}$$

However, the shadow elements are calculated from the language models, rather than directly defined by the language models — hence the definition of $\mathbf{L}^+_\iota$.

This indicates that the paradigm has shifted from a probabilistic definition of SVS parsing towards a vectorized definition. Previously, vectorized definitions were shown to be a valid approximation of pure probabilistic parsing. Here, without first encapsulating the probability distributions in vectors, logical interpretation would not be possible.

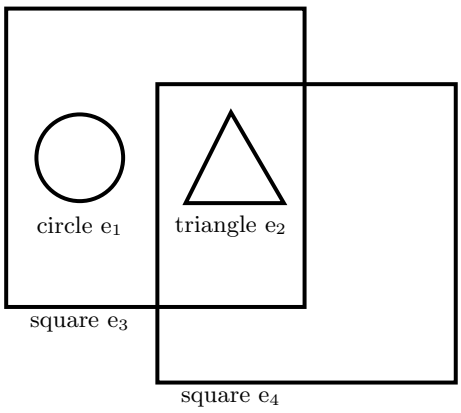
Grammars in θ_{M} in logical interpretation will require lexical information in preterminals. Thus, $\theta_{\text{P-Vit(G)}}$ will produce interesting vectors only if the lexical item denotes a subset of the domain E. Any CNF grammar can be transformed into this type of grammar by multiplying out probabilities for each word in each syntactic relation.

A variety of strategies may be used to determine the *lce*-dependent probabilities — bin them deterministically into entity types, estimate them, or even manually define them. The example in Section 5.4 will take the last approach such that the result will be easily human-readable.

5.4 A Domain-specific Example

This chapter has defined constructs and language models for logical interpretation with SVS, but its scope is only to show that such model-theoretic interpretation is possible. In lieu of giving formal evaluation on a complete system implementing the framework of this chapter, an extended example is provided in this section.

Example (Domain-specific Semantic model)



$$\mathbf{e}_{\text{square}} = \begin{matrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{matrix}\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{L}^{+}_{\text{CONTAIN}} = \begin{matrix} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \end{matrix}\begin{matrix} \begin{matrix} e_1 & e_2 & e_3 & e_4 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{matrix}$$

$$\mathbf{e}_{\text{circle}} = \begin{matrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{matrix}\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{e}_{\text{triangle}} = \begin{matrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{matrix}\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Figure 5.1: Graphical representation of an example world model, defining domain-specific vectors associated with $\theta_{\text{P-Vit(G)}}$ and matrices associated with θ_{L}.

Visually-grounded spoken language interfaces (Gorniak and Roy, 2003; Gorniak and Roy, 2004; Schwartz et al., 2009) often define a world model by spatial and temporal relationships. Figure 5.1 shows a toy graphical domain with four entities which will serve as the indices for vectors and matrices in logical-interpretation SVS. The relationship l_{CONTAINS} is represented by a matrix in which the squares e_3 and e_4 contain the appropriate other objects.

With the world model as shown in Figure 5.1, we will need the full SVS framework to interface this with natural language. Sentences in spoken language interfaces often use the imperative voice to issue commands to an agent. Thus, the following sentence and expected logical interpretation will be used for the running example:

Example (Sentence)

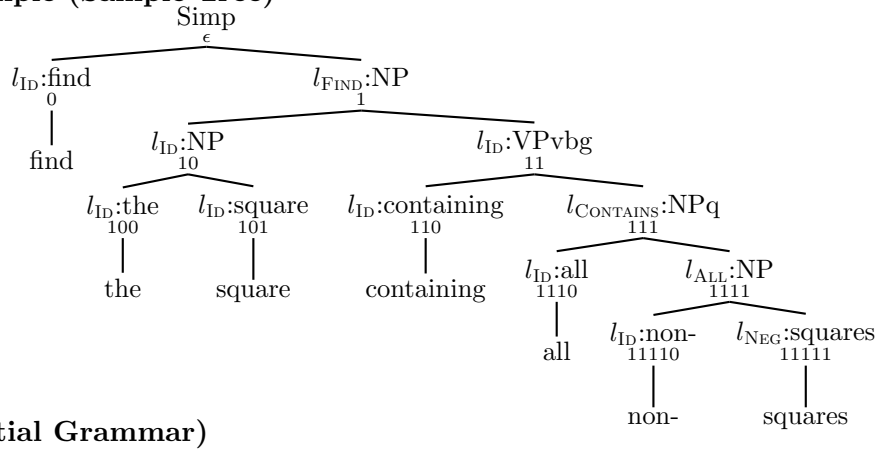
Find the square containing all non- squares.

(Logical Interpretation)

$$\mathbf{e}_\epsilon = e_\epsilon \mapsto \text{Find}(e_\epsilon) \wedge \text{Square}(e_\epsilon) \wedge (\forall e_\eta \mapsto (\neg\text{Square}(e_\eta)) \Rightarrow \text{Contain}(e_\epsilon, e_\eta))$$

The only square containing all non-squares in the domain is e_3; thus, we would expect SVS to produce a vector $\mathbf{e}$ with a non-zero element at e_3.

Example (Sample Tree)



(Partial Grammar)

$\text{Simp}\{e_1,...,e_4\} \rightarrow$	l_{ID}:find	l_{FIND}:NP	$\mathbf{1}$	$= 0.9$	(see ϵ)
$\text{NP}\{e_1,...,e_4\} \rightarrow$	l_{ID}:NP	l_{ID}:VPvbg	$\mathbf{1}$	$= 0.2$	(see 1)
$\text{NP}\{e_3,e_4\} \rightarrow$	l_{ID}:the	l_{ID}:square	$\mathbf{1}$	$= 0.3$	(see 10)
$\text{VPvbg}\{e_3,e_4\} \rightarrow$	l_{ID}:containing	l_{CONTAIN}:NPq	$\mathbf{n}_{\text{TH}}$	$= 0.4$	(see 11)
$\text{NPq}\{e_1,...,e_4\} \rightarrow$	l_{ID}:all	l_{ALL}:NP	$\mathbf{1}$	$= 0.5$	(see 111)
$\text{NP}\{e_1,e_2\} \rightarrow$	l_{ID}:non-	l_{NEG}:squares	$\mathbf{1}$	$= 0.1$	(see 1111)

Figure 5.2: Part of the θ_{M} model, giving grammar rules leading to the production of a sample tree. Other rules and their probabilities are are elided.

In order to interpret this sentence with SVS, let us first assume a particular hypothesized tree — in this case, a very plausible tree as in the first part of Figure 5.2. The addresses for each node have been annotated for further use in this example. Semantic vectors **e** have been left off this tree, leaving only syntactic constituents and semantic relations. How to calculate the semantic vectors will be described later. Assuming that the default task is to 'Find,' the matrix $\mathbf{L}_{\text{FIND}} = \mathbf{I}$; matrices associated with l_{ALL} and l_{NEG} are defined with (distinct) shadow elements as in Section 5.2.

Notice that the relations $l_{\text{CONTAIN}}, l_{\text{ALL}}$, and l_{NEG} are not located at the constituent which subsumes their corresponding text strings; rather, they are placed on the other child. This is because the relations require arguments which are not satisfied by the subsumed word itself, but by the rest of the sentence. In order to allow the relation to operate on the other child, the preterminals are now words; this can be done as a deterministic mapping from some existing CNF source grammar by distributing possible

preterminal probabilities into all possible words.

Figure 5.2 also includes a partial grammar that could have produced the hypothesized tree. Several features of the grammar bear mention. The grammar rules are dependent on individual referents at the parent, indicated by $\{e\}$ — multiple referents imply that there are multiple grammar rules. Not all the referents will have the same syntactic expansion probabilities; NP$\{\mathrm{e}_3, \mathrm{e}_4\}$ is shown as producing 'the square' with probability 0.3, whereas we can assume that NP$\{\mathrm{e}_1, \mathrm{e}_2\}$ will produce 'the square' with zero probability in this domain. This is similar to the reason that lexicalization was originally introduced as a parsing technique, choosing expansions in keeping with the headword.

To account for quantification, the grammar has been specialized in two ways. First, each rule needs to keep track of whether to use a thresholding function $\mathbf{n}_{\text{TH}}$ or a simple vector of ones $\mathbf{1}$. In the case of universal quantification, careful scrutiny of the rules will show that the tresholding operation does not happen at the rule producing the preterminal 'all,' but rather at its parent — which leads to the second adjustment for quantification. The NP class is split into a quantified version, NPq, and a non-quantified version, NP. Then $\mathbf{n}_{\text{TH}}$ is applied in a rule alongside NPq, allowing the quantifier to scope over the binary predicate (l_{CONTAIN} in the example).

Now all the SVS puzzle pieces are in place, and we can see how syntactic-semantic composition is done. To calculate a semantic vector at each tree node, the vector compositions equation (Equation 3.22) is used:

$$\mathbf{e}_\eta = \mathbf{M}(lc_\eta \to lc_{\eta 0}\, lc_{\eta 1}) \cdot \mathrm{d}(\mathbf{L}_{\eta\times\eta 0}(l_{\eta 0}) \cdot \mathbf{e}_{\eta 0}) \cdot \mathrm{d}(\mathbf{L}_{\eta\times\eta 1}(l_{\eta 1}) \cdot \mathbf{e}_{\eta 1}) \cdot \mathbf{1} \tag{3.19′}$$

The semantics-bearing vectors defined by this equation are made specific in Figure 5.3. The figure can be seen to include all the information from the previous tree in Figure 5.2, and this is done completely within the vectors and matrices. Thus, it should be clear that the shift to using vectors and matrices as the canonical structures accomplishes what we would expect.

At preterminals, vectors are named by the word they subsume, and are taken to just be $\mathbf{1}$ except for $\mathbf{e}_{\text{circle}}$, $\mathbf{e}_{\text{triangle}}$, and $\mathbf{e}_{\text{square}}$. Nodes corresponding to preterminals are left out because their Viterbi probabilities are defined by $\theta_{\text{P-Vit(G)}}$ (see the domain in Figure 5.1) rather than any vector or matrix multiplication.

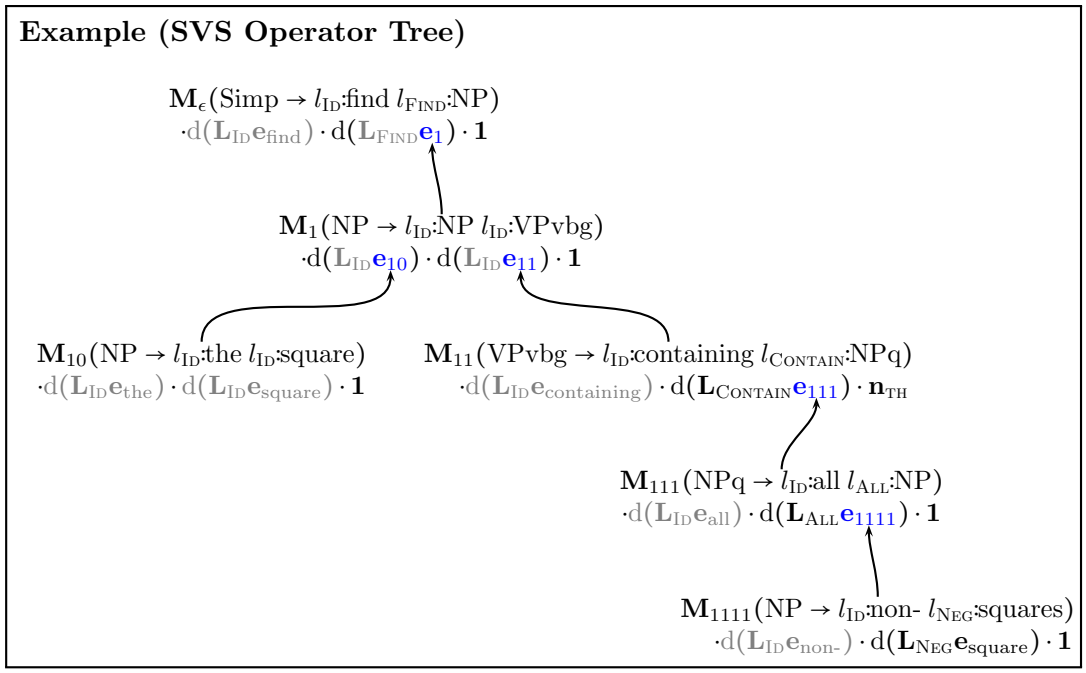


Figure 5.3: SVS operations on a tree, with grammar-rule matrices indicating addresses. Blue variables are defined by their children. Trivial identity operators are shown in gray. Note that the preterminal nodes are not included in this tree.

To simplify the operator tree, the 'Vectors and Matrices' view of Figure 5.4 leaves out grayed variables from Figure 5.3. Also, for $\mathbf{M}_{10}$, the fact that this is a diagonal matrix means that $\text{d}(\mathbf{m}_{10} \cdot \mathbf{1}) = \mathbf{M}_{10}$. The remaining calculations are shown in detail.

The vectors that are substituted into parent equations will usually carry the logical meaning of their subconstituents. Walking up from the bottom constituent, $\mathbf{e}_{\text{square}}$ is seen to select the two squares $\{\text{e}_3, \text{e}_4\}$; $\mathbf{e}_{1111}$ subsumes 'non-square' and therefore selects the other two objects $\{\text{e}_1, \text{e}_2\}$ instead, and the probabilities are from the syntactic rule expansion. Here, $\mathbf{e}_{1111}$ is shown with quantification shadow elements, rather than the negation shadow elements shown in $\mathbf{e}_{\text{square}}$.

Moving on to $\mathbf{e}_{111}$, the same non-squares are still in view, but the probabilities in the quantification shadow elements are crucially different. Because of the l_{ALL} operation, these values are now the sum of the probabilities in the non-shadow part of $\mathbf{e}_{111}$, and will enable the thresholding to work. The input to the thresholding operator is the diagonal matrix $\mathbf{E}_{111} = \text{diag}(0, 0, .2, .2, 0, 0, .2, .4)$.

Example (Vectors and Matrices – Simplified)

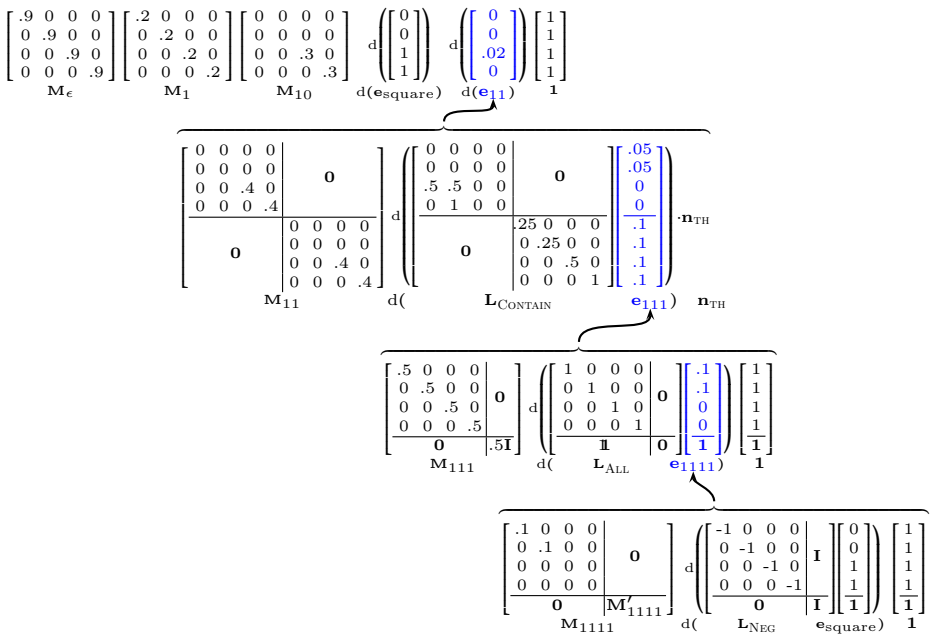


Figure 5.4: A simplified linear operator chain corresponding to the SVS operations tree in Figure 5.3. Shadow elements on the bottom row are different than shadow elements on the second-to-last row.

Moving to the top line, no shadow elements are shown because none of the relations require them. Only e_3 survives when compared to e_3'' in the thresholding function, showing that this is the only entity that contains all the non-squares. This is further conjoined with the set of squares $\mathbf{e}_{\text{squares}}$. The final probability incorporates the syntactic rule expansion probabilities of every step, and selects only the correct referent with corresponding root vector $\mathbf{e}_\epsilon = \begin{bmatrix} 0 & 0 & 0.00108 & 0 \end{bmatrix}$.

5.5 Summary

This chapter has described how Structured Vectorial Semantics could accomplish logical interpretation. Figures 5.1–5.4 demonstrate that logical interpretation with SVS is indeed possible. Future work may include either intricate, manually-defined grammars,

or some type of estimation of how referents and relations interact with each other and with syntax.

Part II

Plausible Language Models with SVS

Chapter 6

Right-Corner Syntactic Parsing

Part I defined Structured Vectorial Semantics on standard PCFGs. It was noted, though, that a CKY-based parser has a language model that assumes psycholinguistically implausible characteristics, such as cubic runtime complexity and a lack of incrementality. An incremental version of SVS would be attractive from an engineering perspective for applications such as real-time speech processing, and it would also constitute a more principled language model.

The next two chapters advance an incremental formulation of SVS that uses the *right-corner transform* in a *Hierarchic Hidden Markov Model*. This first chapter defines both formalisms (Sections 6.1–6.3) and how they jointly accomplish incremental parsing with syntax on right-corner transformed trees (Section 6.4). The definitions will follow Wu et al. (2010), though roughly equivalent parsers have been described thoroughly (Schuler et al., 2008; Schuler et al., 2010).

A different version of the right-corner transform, and its application to an incremental SVS framework, is described in Chapter 7.

6.1 Right-corner transform

Sentences parsed with a Hierarhical Hidden Markov Model will result in trees that have a *right-corner* form. These types of trees are the left–right dual of left-corner trees (Johnson, 1998) and can be mapped to and from ordinary phrase structure via reversible transform rules. Using a grammar built from trees transformed this way, memory usage

in left-to-right traversal of the tree is constrained to a constant bound.

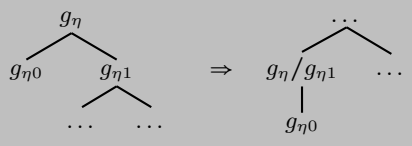
Section 2.3 described a notation for addressing trees and constituents; Figure 6.1 shows where original tree addresses appear before and after a right-corner transform. This is useful because a single transformed tree can simultaneously represent grammatical structure before the right-corner transform (via the subscripted addresses) and after the transform (in the tree structures themselves).

Right-branching sequences in a phrase structure tree[1] transform into left-branching sequences of symbols. Many constituents will be written in the format $g_\eta/g_{\eta\iota}$, denoting an *active* grammar symbol g_η lacking an *awaited* grammar symbol $g_{\eta\iota}$ to the right. This signifies an incremental point in left-to-right traversal, where $g_{\eta\iota}$ is the only unobserved part of g_η.

The right-corner transform may be intuitively understood in 3 cases which are complete in describing the transform and will be used to show the characteristics of right-cornerized trees. The Beginning, Middle, and End of the *originally right-branching structure* are treated differently when they are right-cornerized; we will keep careful track of the address strings to identify when each case applies. To describe these cases, let us temporarily consider η to denote an address string ending with a left branch, '$*0$' (Kleene star '$*$' is used as a wildcard); ι will be a string representing consecutive right branches '1^k'.

The *beginning* of right-branching structure becomes a unary rule, joining the observed left branch $g_{\eta 0}$ with the right branch to be observed. When $g_{\eta 0}$ is observed, g_η has not been observed and is the active constituent; $g_{\eta 1}$ has not been observed and is the awaited constituent.

Right-corner transform: Beginning (of right branch)



(6.1)

Since η is assumed to have a '0' at the end of its string, g_η represents a grammar symbol that was a left child of its parent in the original tree; the beginning of a right branch is

[1] Right-branching structure is where successive right children are regular nonterminals (not preterminals). A tree with such structure expands (branches out) to the right.

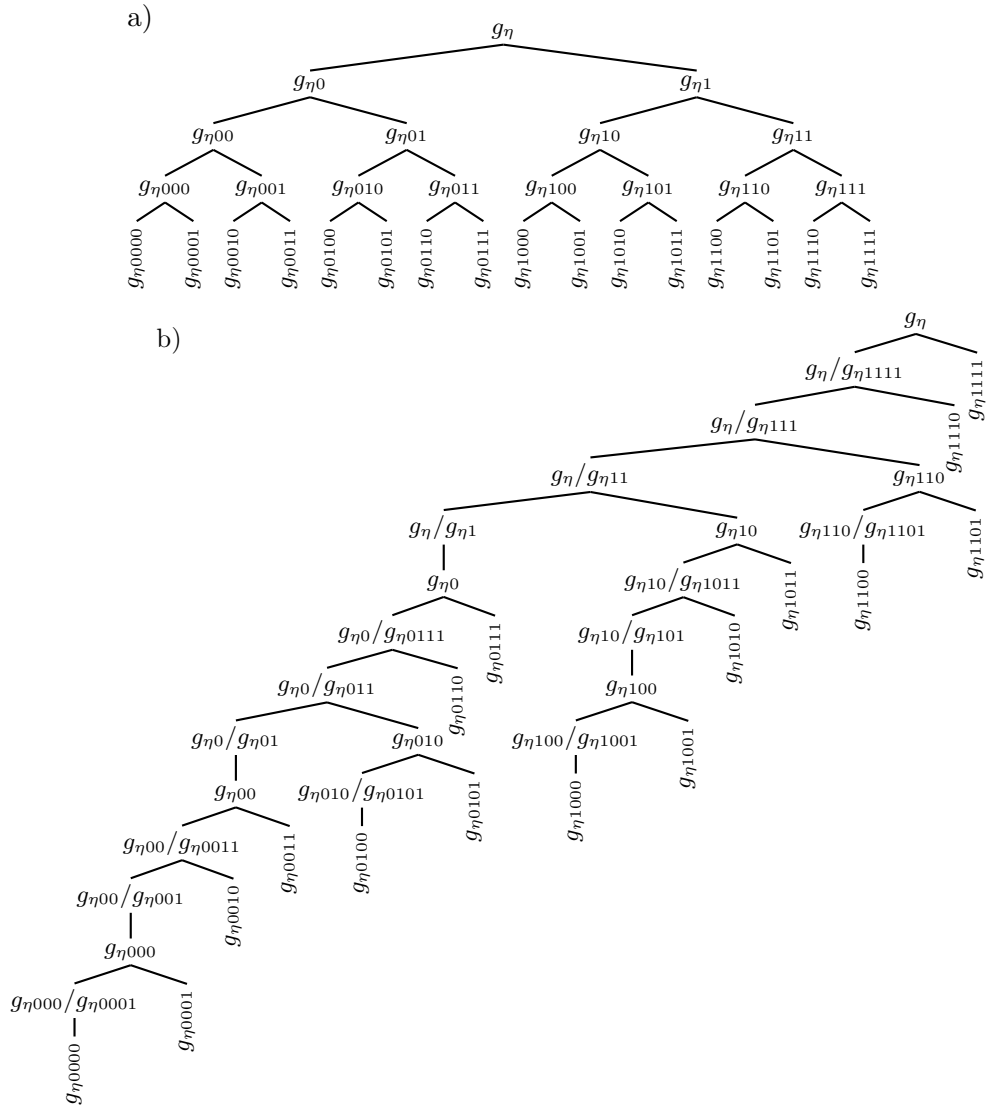
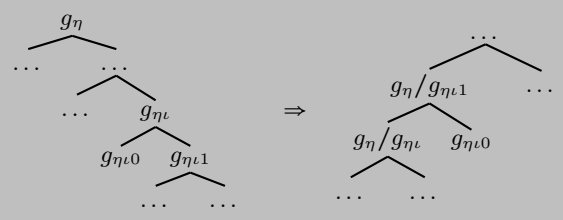


Figure 6.1: a) Normal tree with addresses. b) The same tree, right-corner transformed but with pre-transform addresses.

always itself a left branch. In fact, the Beginning case applies at all left children in the original tree.

Below, a constituent $g_{\eta\iota 0}$ in the *middle* of right-branching structure is one child of the awaited $g_{\eta\iota}$ in $g_\eta/g_{\eta\iota}$, but the other child $g_{\eta\iota 1}$ will still be lacking:

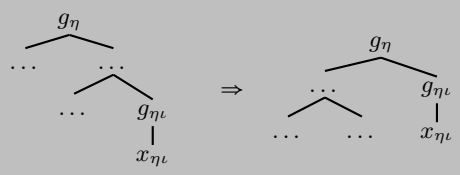
Right-corner transform: Middle (of right branch)



(6.2)

Finally, in the *end* case, a preterminal $g_{\eta\iota}$ at the right corner of right branching structure will be moved to the top level of the transformed (sub)tree along with its terminal symbol, $x_{\eta\iota}$.

Right-corner transform: End (of right branch)



(6.3)

This right corner transform produces trees that are left-branching, with the exception of center-embedded structure in the original tree. *Center-embedding* is any right-then-left zig-zag path, and can be seen in Equation 6.2; if $g_{\eta\iota 0}$ is any non-preterminal constituent then $g_{\eta\iota 0}$ is the beginning of a 'zag.' However, remember that $g_{\eta\iota 0}$ as a subtree will itself be right-cornerized; thus, it will be predominantly left-branching, with the same caveat about center-embedding within itself. But nested center-embedding is relatively infrequent (Schuler et al., 2008).

So if we have a mechanism to parse each (post-transformed) left-branching structure with 1 element of memory (see Section 6.3), the right-corner transform provides a very computationally memory-efficient representation of trees. It organizes trees into the smallest number of left-branching structures possible; these will be referred to later as "levels" or "depths" of a right-corner tree.

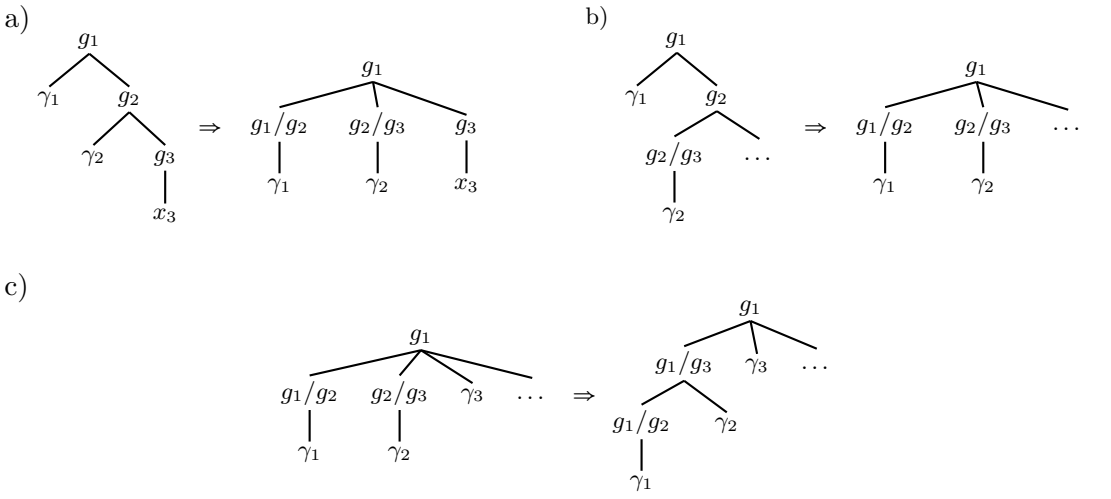


Figure 6.2: Rewrite rules implementing a right corner transform on trees. All g_i denote nonterminal symbols, and all γ_i denote subtrees. Addresses are not shown because all rules are applied recursively, from leaves to root. a,b) remove all right recursion b) replace with left recursion.

The rules above provide the necessary intuition; actually carrying out this transformation on a corpus of binary-branching CNF trees is done as in Schuler, et al. (2008). Formal tree rewrite rules for the right-corner transform are shown in Figure 6.2, first flattenning out right-branching structure (Figure 6.2a,b), then replacing it with left-branching structure (Figure 6.2c).

The first two rewrite rules are applied iteratively (bottom-up on the tree) to flatten all right branching structure, using incomplete constituents to record the original nonterminal ordering. The third rule is then applied to generate left-branching structure, preserving this ordering. Note that the last rewrite above leaves a unary branch at the leftmost child of each flattened node. This preserves the nodes at which the original tree was not right-branching, so the original tree can be reconstructed when the right-corner transform concatenates multiple right-branching sequences into a single left-branching sequence. Rewrite rules for reversing a right-corner transform are simply the converse of those shown above.

6.2 Hidden Markov Models

Hidden Markov Models (HMMs) probabilistically connect sequences of observed states o_t and hidden states s_t at corresponding time steps t. In parsing, observed states are words; hidden states can be a conglomerate state of linguistic information, similar to what might currently be in a person's short-term memory as they process text.

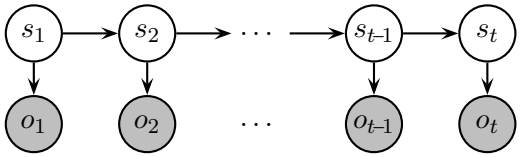


Figure 6.3: Graphical representation of a Hidden Markov Model.

HMMs in parsing assume that from some hidden mental state, you generate an observed word with some probability (downward arrows); moreover, your mental state changes probabilistically as you need to produce different words (horizontal arrows).

A most likely sequence of hidden states $\hat{s}_{1..T}$ can be hypothesized given any sequence of observed states $o_{1..T}$, i.e. words. This relies on a full $\mathsf{P}(s_{1..T} \,|\, o_{1..T})$ probability:

$$\hat{s}_{1..T} = \underset{s_{1..T}}{\operatorname{argmax}} \mathsf{P}(s_{1..T} \,|\, o_{1..T}) \tag{6.4}$$

$$= \underset{s_{1..T}}{\operatorname{argmax}} \mathsf{P}(s_{1..T}) \cdot \mathsf{P}(o_{1..T} \,|\, s_{1..T}) \tag{6.5}$$

$$\overset{\text{def}}{=} \underset{s_{1..T}}{\operatorname{argmax}} \prod_{t=1}^{T} \mathsf{P}_{\theta_\text{A}}(s_t \,|\, s_{t-1}) \cdot \mathsf{P}_{\theta_\text{B}}(o_t \,|\, s_t) \tag{6.6}$$

Bayes' Law is used in producing Equation 6.5 to turn the probabilities into easier-learned generative probabilities (a $\mathsf{P}(o_{1..T})$ term is dropped because it does not affect which $s_{1..T}$ gives the maximum). Equation 6.6 relies on Markov independence assumptions, which constrain the hidden state s_t at any given timestep to only be conditionally dependent on the previous hidden state, s_{t-1}. It also assumes that observations are only dependent on hidden states at the same timestep. In practice, Equation 6.6 is calculated in a process called *inference* or *decoding*, using the Viterbi algorithm to be described later.

What arises from Bayes' Law and the Markov property are[2] a *Transition Model*

[2] Technically, a prior distribution over hidden states, $\mathsf{P}(s_0)$, is necessary. This s_0 is factored and

(θ_{A}) between hidden states

$$\mathsf{P}(s_{1..T}) \stackrel{\text{def}}{=} \prod_t \mathsf{P}_{\theta_{\mathrm{A}}}(s_t \,|\, s_{t\text{-}1}) \tag{6.7}$$

and an *Observation Model* (θ_{B}) that generates an observed model from a hidden state

$$\mathsf{P}(o_{1..T} \,|\, s_{1..T}) \stackrel{\text{def}}{=} \prod_t \mathsf{P}_{\theta_{\mathrm{B}}}(o_t \,|\, s_t). \tag{6.8}$$

These models are termed θ_{A} and θ_{B} for historical reasons (Rabiner, 1990). As a psycholinguistic model, θ_{A} and θ_{B} correspond to changing mental states and the generation of words, respectively.

6.3 Hierarchic Hidden Markov Models

Hidden states s can have internal structure; in Hierarchic HMMs, this internal structure looks like several HMMs stacked on top of each other. So a component HMM within the HHMM would possibly generate other hidden states, rather than generating observations. Some extra overhead is needed to maintain proper synchronization between the different levels.

Transition probabilities $\mathsf{P}_{\theta_{\mathrm{A}}}(s_t \,|\, s_{t\text{-}1})$ over complex hidden states s_t can be calculated in two phases:

- *Reduce phase.* Yields an intermediate state r_t, in which component HMMs may terminate. This r_t tells "higher" HMMs to hold over their information if "lower" levels are in operation at any time step t, and tells lower HMMs to signal when they're done. These variables will be marginalized out in decoding (note the summation in Equation 6.12).

- *Shift phase.* Yields a modeled hidden state s_t, in which unterminated HMMs transition, and terminated HMMs are re-initialized from their parent HMMs.

Variables over intermediate states r_t and modeled states s_t are factored into sequences of depth-specific variables – one for each of D levels in the HMM hierarchy:

$$r_t \stackrel{\text{def}}{=} \langle r_t^1 \ldots r_t^D \rangle \tag{6.9}$$

$$s_t \stackrel{\text{def}}{=} \langle s_t^1 \ldots s_t^D \rangle \tag{6.10}$$

taken to be a deterministic constant, and is therefore unimportant as a probability model.

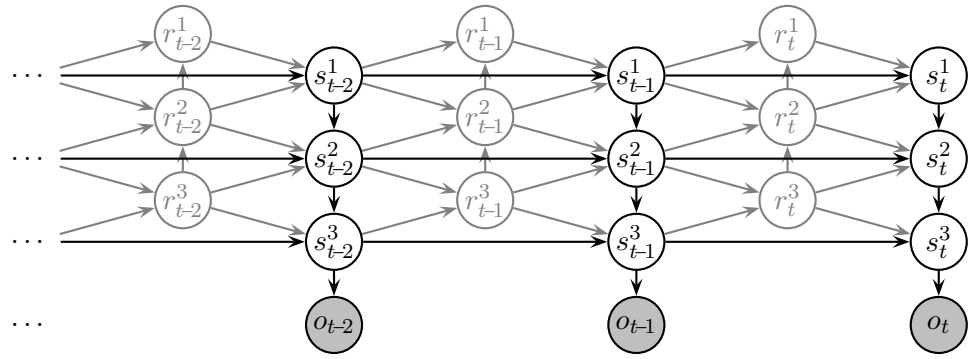


Figure 6.4: Graphical representation of the dependency structure in a Hierarchic Hidden Markov Model with $D=3$ hidden levels that can be used to parse syntax. Circles denote random variables, and edges denote conditional dependencies. Grayed variables are intermediate, and not stored for HHMM inference. Shaded circles denote variables with observed values.

Figure 6.4 shows these depth-specific variables along with their dependency structure, for $D=3$. Directed edges from a source node to a target node indicate that the target variable is conditionally dependent on the source node. If there is no edge, the corresponding variables are assumed to be conditionally independent.

The transition probability incorporating all the levels is calculated as a product of transition probabilities at each level, using level-specific *reduce* θ_{R} and *shift* θ_{S} models:

HHMM Transition Model

$$\mathsf{P}_{\theta_{\mathrm{A}}}(s_t|s_{t-1}) = \sum_{r_t} \mathsf{P}(r_t|s_{t-1}) \cdot \mathsf{P}(s_t|r_t\, s_{t-1}) \tag{6.11}$$

$$\stackrel{\text{def}}{=} \sum_{r_t^{1..D}} \prod_{d=1}^{D} \mathsf{P}_{\theta_{\mathbf{R}}}(r_t^d|r_t^{d+1} s_{t-1}^d s_{t-1}^{d-1}) \cdot \mathsf{P}_{\theta_{\mathbf{S}}}(s_t^d|r_t^{d+1} r_t^d\, s_{t-1}^d s_t^{d-1}) \tag{6.12}$$

with r_t^{D+1} and s_t^0 defined as constants. The dependency structure of Figure 6.4 is entailed by the variables present in the θ_{R} and θ_{S} models. Note that the only variable that is finally present in the probability calculation at each time step is s_t. In step $t+1$, r_t will be unused, so the marginalization of Equation 6.12 does not discard any information.

Following Murphy and Paskin's notation (2001), let us set

$$r_t^d \stackrel{\text{def}}{=} f_t^d \tag{6.13}$$

where f is just a boolean. This r may be defined differently for parsing, e.g., in incremental Structured Vectorial Semantics.

The model probabilities θ_{R} and θ_{S} will now be defined with respect to these boolean variables. The reduction step will possibly reduce a store element (i.e., terminate an HHMM depth) if the store state below it has reduced ($f_t^{d+1}=1$):[3]

HHMM Reduce Step Model

$$\mathsf{P}_{\theta_{\mathrm{R}}}(r_t^d \mid r_t^{d+1} s_{t-1}^d s_{t-1}^{d-1}) \stackrel{\text{def}}{=} \begin{cases} \text{if } f_t^{d+1}=1 : \mathsf{P}_{\theta_{\text{R-Rdn},d}}(r_t^d \mid r_t^{d+1} s_{t-1}^d s_{t-1}^{d-1}) \\ \text{if } f_t^{d+1}=0 : [\![r_t^d = \mathrm{r}_\perp]\!] \end{cases} \tag{6.14}$$

where $s_t^0 = \mathrm{s}_\top$ and $r_t^{D+1} = \mathrm{r}_\top$ for constants $\mathrm{s}_\top$, $\mathrm{r}_\top$, and $\mathrm{r}_\perp$ (a null state). The second case is simply a copy operator, occuring whenever lower HHMM depths are incomplete. This indicates that the node is above the "frontier" of action, waiting for some state that it had previously generated to complete before it will itself complete.

The shift step in HHMMs also have a copy operator when they are not on the "frontier" (the third case below), but are more interesting due to the first two cases:

HHMM Shift Step Model

$$\mathsf{P}_{\theta_{\mathrm{S}}}(s_t^d \mid r_t^{d+1} r_t^d \; s_{t-1}^d s_t^{d-1}) \stackrel{\text{def}}{=} \begin{cases} \text{if } f_t^{d+1}=1,\ f_t^d=1 : \mathsf{P}_{\theta_{\text{S-Exp},d}}(s_t^d \mid s_t^{d-1}) \\ \text{if } f_t^{d+1}=1,\ f_t^d=0 : \mathsf{P}_{\theta_{\text{S-Trn},d}}(s_t^d \mid r_t^{d+1} r_t^d \; s_{t-1}^d s_t^{d-1}) \\ \text{if } f_t^{d+1}=0,\ f_t^d=0 : [\![s_t^d = s_{t-1}^d]\!] \end{cases} \tag{6.15}$$

The first case starts a new level in the HHMM by expanding to (i.e., generating) a new incomplete constituent; this only happens after a reduction has taken place ($f_t^d=1$). The second transitions along a sequence of store elements on the same level if no reduction has taken place ($f_t^d=0$).

The Observation Model θ_{B} is comparatively much simpler. It is only dependent on the hidden state at D (i.e., the deepest active HHMM stack level). In practice, it will not even use the whole hidden state, but only the most relevant syntactic (the awaited syntactic category, as will be described later).

[3] Recall that $[\![\cdot]\!]$ is an indicator function: $[\![\phi]\!] = 1$ if ϕ is true, 0 otherwise.

HHMM Observation Model

$$\mathsf{P}_{\theta_{\mathrm{B}}}(o_t \mid s_t) \stackrel{\text{def}}{=} \mathsf{P}(o_t \mid s_t^D) \tag{6.16}$$

What would seem to be an inherent limitation of this model is its need to specify the number of depth levels, D. However, according to the literature (Miller and Chomsky, 1963; Cowan, 2001), the human language faculty can only employ a finite number of short-term memory elements — and this is the memory that is likely used for syntactic processing. Previous work has shown that most sentences in English can be parsed with 4 or fewer depth-levels (Schuler et al., 2008; Schuler et al., 2010). Thus, evaluations in this paper are done with $D = 4$.

An HHMM defined with the Reduce and Shift steps as above will store a probability at each modeled hidden state s_t corresponding to the best path arriving at that particular state. This probability can be written as a recurrence and calculated from hypotheses at the previous time step.

$$\begin{aligned}\mathsf{P}(x_{1..t}, s_t^{1..D}) = \max_{s_{t-1}^{1..D}} & \mathsf{P}(x_{1..t-1}, s_{t-1}^{1..D}) \\ & \cdot \max_{r_t^1 .. r_t^D} \prod_d \mathsf{P}_{\theta_{\mathrm{R}}}(r_t^d | r_t^{d+1} s_{t-1}^d s_{t-1}^{d-1}) \cdot \mathsf{P}_{\theta_{\mathrm{S}}}(s_t^d | r_t^{d+1} r_t^d \, s_{t-1}^d s_t^{d-1}) \\ & \cdot \mathsf{P}_{\theta_{\mathrm{B}}}(x^{\mathrm{W}}_{s_t^D} \mid c^{\mathrm{W}}_{s_t^D})\end{aligned} \tag{6.17}$$

The first line presents the probability at states in the previous time step, the second line calculates the transition probabilities to get to the current step, and the third line applies the observation probabilities. The resulting joint probability is used to rank competing hypothesized sequences. At time T, the probability corresponds to a whole sequence of modeled states that define a complete tree. This recurrence equation will be revisited in Chapter 7.

6.4 Right-corner trees and HHMMs

HHMM states and dependencies, as defined above, can be optimized for the parsing of right-corner trees (Schuler et al., 2008; Schuler et al., 2010). Figure 6.5 gives an example tree that would first be recognized in right-corner form by the HHMM, then reverse-transformed into CNF form for evaluation.

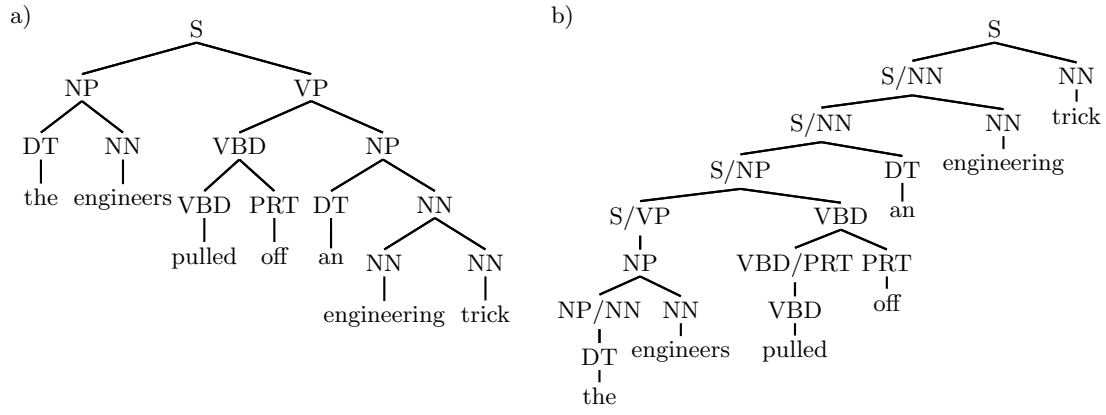


Figure 6.5: a) A tree in CNF. b) The same tree after a right-corner transform.

For an intuition of the mapping from right-corner trees to HHMMs, recall the right-corner transformation's effect on syntactic constituents. The overall shape of a right-corner tree is dominated by left-branching "trunks." New trunks are only created when the original tree had center-embedded structure, defined as right-then-left ('10') address paths — e.g., the binary-branching VBD on Figure 6.5a, which follows a right-then left path from its parent. The HHMM parser will represent each of these trunks as a separate random variable (i.e. a memory element) at each time step, and bound the number of trunks that can be considered.

It should be mentioned that during recognition of this right-corner tree, a left-to-right traversal will require that the new level be started *without the subsequent information on that level*. Thus, the preterminal VBD would be hypothesized to generate the word "pulled" before knowing that its parent was actually a VBD as well. This indicates that some estimation will be necessary to recognize right-corner trees incrementally. It also means that the two levels will not be fully attached to each other until the lower level is complete.

In Figure 6.6, we introduce a stack interpretation[4] of the right corner tree in Figure 6.5. The correspondence to both the HHMM dependency structure (Figure 6.4) and the right-corner tree (Figure 6.5b) should be clear.

[4] This is technically a pushdown automoton (PDA), where the stack is limited to D elements. When referring to directions (up, down, higher, lower), PDA stacks are typically described opposite of the one pictured in Figure 6.6; the stack here pushes "up" instead of down.

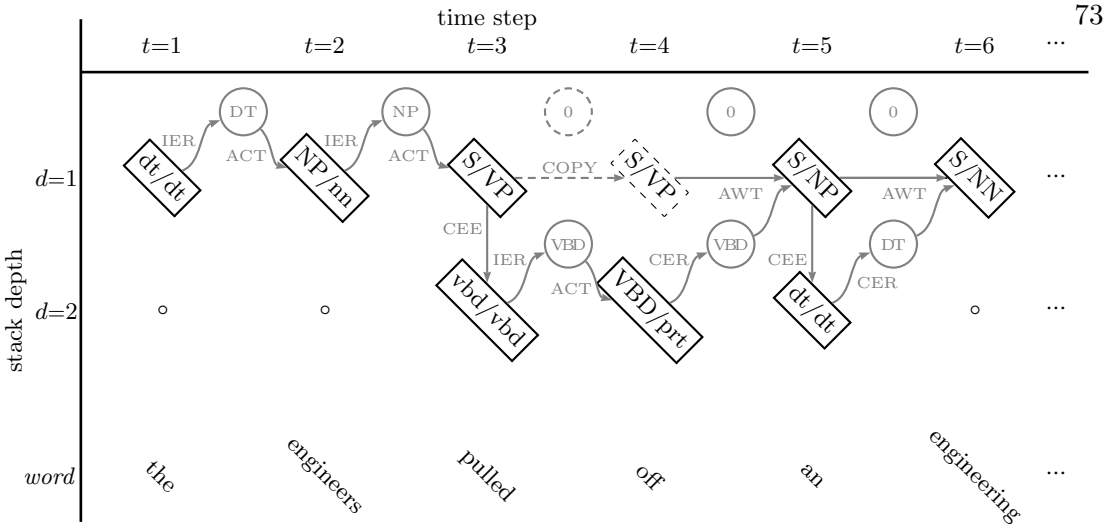


Figure 6.6: Sequence of HHMM store states for a sample sentence "The engineers pulled off an engineering trick," showing a good syntactic hypothesis out of many kept in parallel. Variables corresponding to s_t^d are boxed; r_t^d are circled. Right-corner parsing operations are labeled on the arrows.

Each modeled state contains the active and awaited constituent categories ($c^{\mathrm{A}}_{s_t^d}$ and $c^{\mathrm{W}}_{s_t^d}$):

$$s_t^d \stackrel{\text{def}}{=} \langle c^{\mathrm{A}}_{s_t^d}, c^{\mathrm{W}}_{s_t^d} \rangle \tag{6.18}$$

and each reduction state contains the nearest center-embedded (right-then-left child) complete constituent category $c_{r_t^d}$, as well as a flag $f_{r_t^d}$ indicating when a right corner has been reached in the tree:

$$r_t^d \stackrel{\text{def}}{=} \langle f_{r_t^d}, c_{r_t^d} \rangle \tag{6.19}$$

At each time step t, a certain number of elements (maximum D) are kept in memory, i.e., in the stack. A new word is introduced as observed input at each time step, and the bottom occupied element (the stack "frontier") is the context; together, this word and frontier element determine what the stack will look like at $t{+}1$.

There is one unique set of HHMM state values for each hypothesized tree, so the operations can be seen on either the tree or the stack. Note that s_t^d grammatical constituents lie along the left-branching trunk of the tree and are always incomplete (containing a '/').[5] The helper variables r_t^d are shown in grayed circles and are complete

[5] Lower-cased incomplete constituents, e.g. dt/dt in Figure 6.6, are deterministically mapped from

constituents — they are always the child of some unary rule in the post-transformed tree.

We can characterize the types of stack-element changes by when they happen in Figures 6.5b and 6.6:

- *Cross-element Expansion* (CEE). Starts a new stack level at a given time step. On a tree, this is the leftmost corner of a new left-branching trunk.

- *In-element Reduction* (IER). Completes an active constituent that is a unary child in the right-corner tree; *always* followed by an in-element expansion. Comes from a left branch in the original tree.

- *Active Transition* (ACT). Also known as an in-element expansion, this starts a new active constituent at an already-active stack level; always preceded by an in-element reduction. Together with an IER, this accounts for unary branches along left-branching trunks in post-transformed trees.

- *Awaited Transition* (AWT). Also known as an in-element transition, this transitions the stack to a new state in the next time step where the incomplete active constituent is the same but the awaited constituent has changed. This occurs for binary rules along the post-transformed left-branching trunk.

- *Cross-element Reduction* (CER). Terminates a stack level on seeing a complete active constituent, namely, the top constituent of a trunk level in the post-transformed tree, which will be a right child of the level above it; always followed by an awaited transition at the level above.

HHMM Reduce and Shift models from Equations 6.14–6.15 may then be refined for the stack-element changes of right-corner parsing. Here, active transitions $\theta_{\text{G-ACT},d}$ are distinguished from awaited transitions $\theta_{\text{G-AWT},d}$, and the non-frontier reduce elments are effectively ignored:

$$\mathsf{P}_{\theta_{\text{S-Exp},d}}(s_t^d \mid s_t^{d-1}) \stackrel{\text{def}}{=} \mathsf{P}_{\theta_{\text{G-CEE},d}}(s_t^d \mid s_t^{d-1}) \tag{6.20}$$

$$\mathsf{P}_{\theta_{\text{S-Trn},d}}(s_t^d \mid r_t^{d+1} r_t^d \, s_{t-1}^d s_t^{d-1}) \stackrel{\text{def}}{=} \begin{cases} \text{if } r_t^d = \text{r}_\perp : \mathsf{P}_{\theta_{\text{G-AWT},d}}(s_t^d \mid s_{t-1}^d r_t^{d+1}) \\ \text{if } r_t^d \neq \text{r}_\perp : \mathsf{P}_{\theta_{\text{G-ACT},d}}(s_t^d \mid s_t^{d-1} r_t^d) \end{cases} \tag{6.21}$$

the unary branching DT in Figure 6.5b. This is the CEE case.

$$\mathsf{P}_{\theta_{\text{R-Rdn},d}}(r_t^d \mid r_t^{d+1} s_{t-1}^d s_{t-1}^{d-1}) \stackrel{\text{def}}{=} \begin{cases} \text{if } c_{r_t^{d+1}} = x_t : \mathsf{P}_{\theta_{\text{G-CER},d}}(r_t^d \mid s_{t-1}^d s_{t-1}^{d-1}) \\ \text{if } c_{r_t^{d+1}} \neq x_t : [\![r_t^d = \text{r}_\perp]\!] \end{cases} \tag{6.22}$$

The reduction case has only a $\theta_{\text{G-CER},d}$ model the probabilistic dependencies for both IER and CER are the same; the target value $f_{r_t^d} = \{0, 1\}$ determines which case it falls under.

Probability models for right-corner parsing with HHMMs can be empirically determined directly from right-corner transformed trees, based on these 5 cases. This is the strategy used in the majority of previous work with right-corner HHMM parsing (Schuler et al., 2008; Schuler et al., 2010; Wu, Schwartz, and Schuler, 2008b).

The intuitive mapping between right-corner trees (Figure 6.5b) and the HHMM stack (Figure 6.6) may be formally specified. Below, empirical frequencies are written as $\tilde{\mathsf{P}}(\cdot)$, and conditioned frequencies are obtained by dividing by marginalized distributions.

Cross-element expansions (CEE) occur when center-embedding structure begins in the original tree. In other words, a right child is followed by k left-branching rules to create a tree. Thus, the probability model can be determined by:

$$\mathsf{P}_{\theta_{\text{G-CEE},d}}(s_t^d \mid s_t^{d-1}) \stackrel{\text{def}}{=} \tilde{\mathsf{P}}(\langle c_{\eta\iota}, c'_{\eta\iota} \rangle \mid \langle -, c_\eta \rangle) \tag{6.23}$$

where $\eta = *1$ (any right child in the original grammar), $\iota = 0^k$ (a string of left-expansions), and '$-$' means that the probability distribution marginalizes over that active constituent. The restriction on strings is important, and can be seen in Figure 6.1.

For awaited transitions (AWT):

$$\mathsf{P}_{\theta_{\text{G-AWT},d}}(s_t^d \mid s_{t-1}^d r_t^{d+1}) \stackrel{\text{def}}{=} \tilde{\mathsf{P}}(\langle c_\eta, c_{\eta\iota 1} \rangle \mid \langle c'_\eta, c_{\eta\iota} \rangle \; c_{\eta\iota 0}) \tag{6.24}$$

with string definitions $\eta = *0$ (a left child in the original grammar) and $\iota = 1^k$ (a string of right-expansions in the original grammar).

For active transitions (ACT):

$$\mathsf{P}_{\theta_{\text{G-ACT},d}}(s_t^d \mid s_t^{d-1} r_t^d) \stackrel{\text{def}}{=} \tilde{\mathsf{P}}(\langle c_{\eta\iota}, c_{\eta\iota 1} \rangle \mid \langle -, c_\eta \rangle \; c_{\eta\iota 0}) \tag{6.25}$$

with $\eta = *1$ (a right child in the original grammar) and $\iota = 0^k$ (left-branch in the original grammar). Note that $c_{\eta\iota}$ is assumed to have children $c_{\eta\iota 0}$ and $c_{\eta\iota 1}$; in practice, it is easy to distinguish between CEE and ACT despite the necessary indices being similar.

Cross-element reductions (CER) are a special case of $\theta_{\text{G-CER},d}$ where the modeled variable r_t^d has $f_{r_t^d} = 1$:

$$\mathsf{P}_{\theta_{\text{G-CER},d}}(r_t^d \mid s_{t-1}^d s_{t-1}^{d-1}) \stackrel{\text{def}}{=} \tilde{\mathsf{P}}(c_{\eta 0}, 1 \mid \langle c'_{\eta 0}, - \rangle \, \langle -, c_\eta \rangle) \tag{6.26}$$

Here, $\eta = *1$ (a right child); thus, $\eta 0$ is the complete constituent associated with the completing stack level.

In-element reductions (IER) are another case of $\theta_{\text{G-CER},d}$, where r_t^d has $f_{r_t^d} = 0$:

$$\mathsf{P}_{\theta_{\text{G-CER},d}}(r_t^d \mid s_{t-1}^d s_{t-1}^{d-1}) \stackrel{\text{def}}{=} \tilde{\mathsf{P}}(c_{\eta \iota}, 0 \mid \langle c'_{\eta \iota}, - \rangle \, \langle -, c_\eta \rangle) \tag{6.27}$$

and $\eta = *1$ as in the CER case, but now $\iota = 0^k$ (CER could be considered to have $\iota = 0$). This covers any of the unary rules along a right-corner transformed 'trunk.'

Counting these frequencies on right-corner transformed trees gives HHMM probabilities tuned specifically for right-corner parsing. Schuler, et al. (Schuler et al., 2010) report an F-score of 77.7 on Section 23–24 of the Wall Street Journal with no punctuation, using this type of strategy. This is comparable to the parsing performance of the baseline CKY parsers as described in Chapter 4 (most of the difference is accounted for by the absence of punctuation).

6.5 Summary

This chapter has provided the necessary background for incremental structured vectorial semantics, introducing the tree-based right-corner transform and Hierarchical Hidden Markov Models. The right-corner transform naturally represents an incremental account of syntax. This incremental structure is laid onto a stack interpretation of HHMMs, such that the probabilities for right-corner syntactic HHMM parsing can be learned from trees.

Chapter 7

Incremental Structured Vectorial Semantics

This chapter presents a language model that bears a number of psycholinguistically plausible characteristics in addition to the text comprehension assumptions that are implicit in SVS. One of these characteristics is *incrementality*, a trait implicit in the use of an HHMM parser to recognize a right-corner transformed version of SVS. Incremental vectorial semantic composition interacts with incremental syntactic parsing in this language model. In addition, a *bounded store* of short-term memory, and ensuing difficulty in language comprehension, is implied through the HHMM framework. Detailed treatment of the language modeling principles incorporated in this multifaceted incremental SVS language model will be described later (Chapter 8).

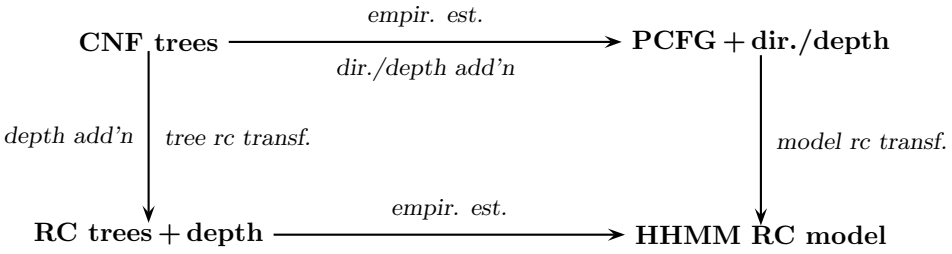
Of the chapters in this thesis, this is the most technical. Therefore, thoroughly understanding SVS (Chapter 3) and HHMM parsing (Chapter 6) is a prerequisite, since the ensuing discussion will combine these two frameworks.

Also, the familiarity with logical-interpretation instantiation of SVS (Chapter 5) is assumed for discussion on linguistic predictions of incremental SVS. The non-linear thresholding operator necessary in SVS's definition of universal quantification will place strains on the incremental SVS model of short-term memory; this means that universal (and other non-existential) quantifiers should occur less frequently in syntactic structures where more short-term memory is in use.

The chapter will begin by providing an alternate form of the work of Schuler (2009), which describes a model-based (rather than tree-based) right-corner transform, and applies this transform for syntactic HHMM parsing (Section 7.1). Then, this alternate right-corner transform formulation will be used to incrementalize SVS for HHMM parsing (Section 7.2); finally, linguistic predictions that fall out of the incremental SVS model with logical interpretation will be considered (Section 7.3).

7.1 Incremental Syntactic Parsing with HHMMs

Rather than performing the right-corner transform on trees, it is also possible to perform the transform on PCFG models (Schuler, 2009). Because Structured Vectorial Semantics was specified on PCFGs and not on trees, this will be crucial in defining SVS in incremental form. A model-based transformation will be based on the empirically-determined grammar models, and will produce probability models that estimate left-to-right traversal of right-corner trees.



As can be seen in the figure above, there are both similarities and differences between the over-then-down (model-based) steps to get to a right-corner HHMM parsing model and the down-then-over (tree-based) right-corner HHMM parsing. Both methods arrive at HHMM equations that parse input strings into right-corner trees. The resulting HHMM rules for both are specified by *center-embedding depth*, counted as how many right-then-left-child paths are traversed to get to a node. The center-embedding depth is limited to a number D (in evaluations for previous work, $D = 4$).

However, in the over-then-down PCFG model strategy, *direction* — whether the parent constituent is a left or right child of its own parent — must also be included. Both the direction and depth can be added either before or after the empirical estimation step. In other words, in the diagram above, the operations above and below the top

line may be done in either order. Doing the empirical estimation first (Schuler, 2009) yields a standard PCFG (without direction or depth) as an intermediate state, upon which direction and depth are later estimated. Doing direction and depth annotation first (assumed in this chapter) yields augmented trees as an intermediate step, and depth-limited grammar rules are estimated afterwards.

The PCFG probabilities in the top right of the figure are $\theta_{\text{G-L},d}$ and $\theta_{\text{G-R},d}$, in place of a standard PCFG θ_{G}. These direction-/depth-specific grammars are more sparse than those without these specifications, but they been shown to parse with negligible accuracy differences.

With a direction/depth-specific PCFG, how should a right-corner parser be defined? Chapter 6 showed that the right-corner transform on trees treated the Beginning, Middle, and End of right-branching structure as separate cases (see Equation 6.1 and following on page 63). So we may expect a model-based right-corner transform to be similar.

Let us begin by observing that any prefix of a yield x_η can be rewritten as an 'incomplete yield,' consisting of the complete yield lacking some suffix of that yield $x_{\eta\iota}$:

$$x_\eta = (x_\eta - x_{\eta\iota}) \circ x_{\eta\iota} \tag{7.1}$$

It is therefore possible to decompose any Viterbi probability into two parts: first, an 'incomplete constituent' probability of generating this incomplete yield $x_\eta - x_{\eta\iota}$ along with an *awaited* constituent of category $c_{\eta\iota}$, given an *active* constituent of category c_η; and second, an ordinary Viterbi probability (or 'complete constituent' probability) of generating $x_{\eta\iota}$ given $c_{\eta\iota}$.

If we specify that $\eta\iota$ is the eventual right corner below address η, then this alternate decomposition corresponds with the End case of the tree-based right-corner transform (Equation 6.3):

$$\mathsf{P}_{\theta_{\text{Vit(G)},d}}(x_\eta \,|\, c_\eta) = \max_{\substack{\iota,\ c_{\eta\iota} \\ x_{\eta\iota}}} \mathsf{P}_{\theta_{\text{IC(G)},d}}(x_\eta - x_{\eta\iota},\ c_{\eta\iota} \,|\, c_\eta) \cdot \mathsf{P}_{\theta_{\text{P-Vit(G)}}}(x_{\eta\iota} \,|\, c_{\eta\iota}) \tag{7.2}$$

Here, η is an arbitrary string, $\iota = 1^k$ is the repeated right branches until the right corner, and $x_{\eta\iota}$ is the last word in the yield x_η. Maximizing over x values does not mean all possible words in the dictionary. Rather, they are yield spans that define how much of the observed input is subsumed by a particular address. In this case, by specifying that

the last term is a preterminal probability from $\theta_{\text{P-Vit(G)}}$, we ensure that only the right corner (last word) in the original subtree (substring) is subsumed, and the maximum over spans is trivial. However, the maximum over ι is not trivial, since the last word could be located anywhere along the right branch from the root.

The resulting incomplete constituent probabilities can then be decomposed into other incomplete constituent probabilities, which is analogous to the tree-based Middle case (Equation 6.2):

$$\begin{aligned}
&\mathsf{P}_{\theta_{\text{IC(G)},d}}(x_\eta{-}x_{\eta\iota 1},\ c_{\eta\iota 1} \,|\, c_\eta) \\
&\quad = \max_{c_{\eta\iota}} \mathsf{P}_{\theta_{\text{IC(G)},d}}(x_\eta{-}x_{\eta\iota},\ c_{\eta\iota} \,|\, c_\eta) \cdot \mathsf{P}_{\theta_{\text{IC(G)},d}}(x_{\eta\iota}{-}x_{\eta\iota 1},\ c_{\eta\iota 1} \,|\, c_{\eta\iota}) && (7.3) \\
&\quad = \max_{c_{\eta\iota}} \mathsf{P}_{\theta_{\text{IC(G)},d}}(x_\eta{-}x_{\eta\iota},\ c_{\eta\iota} \,|\, c_\eta) \cdot \max_{\substack{c_{\eta\iota 0} \\ x_{\eta\iota 0}}} \mathsf{P}_{\theta_{\text{G-R},d}}(c_{\eta\iota} \rightarrow c_{\eta\iota 0}\ c_{\eta\iota 1}) \cdot \mathsf{P}_{\theta_{\text{Vit(G)},d+1}}(x_{\eta\iota 0} \,|\, c_{\eta\iota 0}) && (7.4)
\end{aligned}$$

where again, $\iota = 1^k$.

Incomplete constituent probablities can also be decomposed into the product of a grammar rule probability and a Viterbi probability at the end of a sequence of such decompositions; this corresponds to the Begin case (Equation 6.1):

$$\mathsf{P}_{\theta_{\text{IC(G)},d}}(x_\eta{-}x_{\eta 1},\ c_{\eta 1} \,|\, c_\eta) = \max_{\substack{c_{\eta 0} \\ x_{\eta 0}}} \mathsf{P}_{\theta_{\text{G-L},d}}(c_\eta \rightarrow c_{\eta 0}\ c_{\eta 1}) \cdot \mathsf{P}_{\theta_{\text{Vit(G)},d}}(x_{\eta 0} \,|\, c_{\eta 0}) \qquad (7.5)$$

Essentially, this turns right-expanding sequences of constituents (with subscript addresses ending in '1') into left-expanding sequences of incomplete constituents, which can be composed together as they are recognized incrementally from left to right.

The decompositions of Equations 7.2–7.5, taken together, describe the probabilities that would be calculated by a bottom-up right-corner parser at each depth level. In this formulation, connections between depth levels are made exclusively in the final term of Equation 7.4. Thus, the incremental probabilities and completed Viterbi probabilities are *local*.

At the root of the tree, $\eta = \epsilon$ in Equation 7.2, $d = 1$ with no other depth levels interacting, and the last input symbol is being observed. This works as a bottom-up parsing strategy, but how would analyses with multiple incomplete depth levels look, incrementally? Because these $\theta_{\text{IC(G)},d}$ are local to their depth levels, they are not

attached to each other in incremental recognition. The attachments would be made clear at the end, but it will be helpful to approximate these connections between depth levels at each time step (and remove those approximations when the real attachment is observed). This way, each time step has a reasonable set of ranked hypotheses that estimate the most likely incremental 'attached' tree. For a framework like the HHMM, this is so that good final analyses are less likely to fall off the HHMM beam.

Therefore, the model $\theta_{\text{G-RL}^*,d}$ will contain expected frequency counts over possible expansions in the grammar; expected counts help rules weigh their relative importance in the HHMM. Expected counts for arbitrary numbers of left-branches (following the right branch) are calculated inductively:

$$\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{0}{\rightarrow} c_{\eta 0}\ \ldots) = \sum_{c_{\eta 1}} \mathsf{P}_{\theta_{\text{G-R},d}}(c_\eta \rightarrow c_{\eta 0}\ c_{\eta 1}) \tag{7.6}$$

$$\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{k}{\rightarrow} c_{\eta 0^k 0}\ \ldots) = \sum_{c_{\eta 0^k}} \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{k-1}{\rightarrow} c_{\eta 0^k}\ \ldots) \sum_{c_{\eta 0^k 1}} \mathsf{P}_{\theta_{\text{G-L},d}}(c_{\eta 0^k} \rightarrow c_{\eta 0^k 0}\ c_{\eta 0^k 1}) \tag{7.7}$$

$$\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{*}{\rightarrow} c_{\eta\iota}\ \ldots) = \sum_{k=0}^{\infty} \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{k}{\rightarrow} c_{\eta\iota}\ \ldots) \tag{7.8}$$

$$\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{+}{\rightarrow} c_{\eta\iota}\ \ldots) = \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{*}{\rightarrow} c_{\eta\iota}\ \ldots) - \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{0}{\rightarrow} c_{\eta\iota}\ \ldots) \tag{7.9}$$

In practice the infinite sum is estimated to some constant K using value iteration (Bellman, 1957).

Where might this be used in an HHMM to parse right-corner trees? For some intuition, recall that the CEE case expanded an awaited constituent c_η to a constituent $c_{\eta\iota}$ on a new depth level, where $\eta = *1$ and $\iota = 0^k$. In a tree, both of these constituents are simultaneously available (since all rule expansions are known) during training, but in a PCFG, only one rule can be considered at a time. Therefore, there is need to consider the possibility of an arbitrary number of left-expansions after a right branch via the model $\theta_{\text{G-RL}^*,d}$. The in-element expansion ACT was defined very similarly, and also uses $\theta_{\text{G-RL}^*,d}$.

The '0-repeated' expansion and the '+-repeated' expansion versions of $\theta_{\text{G-RL}^*,d}$, calculated as part of the induction, are useful in the reduction cases CER and IER. The cross-element case only considers a single left-expansion, whereas the in-element case considers all left-expansions beyond the first.

The HHMM formulation of the right-corner transform will give formally equivalent

results to Equations 7.2–7.5; however, it will do Viterbi 'max' operations in different locales and utilize the estimated $\theta_{\text{G-RL}^*,d}$ rules to connect between multiple levels of depth-specific incomplete constituents.

Consider the below equivalent representations for the probability at a modeled hidden state.

$$\mathsf{P}(x_{1..t}, s_t^{1..D})$$

$$= \left[\sum_{s_{t-1}^{1..D}} \prod_d \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c^{\text{W}}_{s_t^{d-1}} \overset{*}{\rightarrow} c^{\text{A}}_{s_t^d} \ldots) \cdot \mathsf{P}_{\theta_{\text{IC(G)},d}}(x^{\text{A}}_{s_t^d} \text{–} x^{\text{W}}_{s_t^d}, c^{\text{W}}_{s_t^d} \mid c^{\text{A}}_{s_t^d}) \right] \cdot \mathsf{P}_{\theta_{\text{P-Vit(G)}}}(x^{\text{W}}_{s_t^D} \mid c^{\text{W}}_{s_t^D}) \tag{7.10}$$

$$= \left[\prod_{\tau=1}^{t-1} \mathsf{P}_{\theta_{\text{A}}}(s_\tau \mid s_{\tau-1}) \cdot \mathsf{P}_{\theta_{\text{B}}}(x_\tau \mid s_\tau) \;\cdot\; \mathsf{P}_{\theta_{\text{A}}}(s_t \mid s_{t-1}) \right] \cdot \mathsf{P}_{\theta_{\text{B}}}(x^{\text{W}}_{s_t^D} \mid c^{\text{W}}_{s_t^D}) \tag{7.11}$$

$$= \left[\max_{s_{t-1}^{1..D}} \mathsf{P}(x_{1..t-1}, s_{t-1}^{1..D}) \;\cdot\; \max_{r_t^1 .. r_t^D} \prod_d \mathsf{P}_{\theta_{\text{R}}}(\cdots) \cdot \mathsf{P}_{\theta_{\text{S}}}(\cdots) \right] \cdot \mathsf{P}_{\theta_{\text{B}}}(x^{\text{W}}_{s_t^D} \mid c^{\text{W}}_{s_t^D}) \tag{7.12}$$

First, a modeled depth-specific state at each level of the HHMM hierarchy represents a local incomplete constituent with incomplete probabilities[1] IC(G),d and estimated cross-level attachments from $\theta_{\text{G-RL}^*,d}$. This is equivalent to (the second equation) a step-by-step multiplication of the transition model θ_{A} producing all yields except the current one. For the final equality, recall from Equation 6.17 that the probability of a modeled hidden state s_t can be written as a recurrence (θ_{R} and θ_{S} dependencies are collapsed for brevity).

Notice that the observation model θ_{B} is present in every equation; this probability will always be a preterminal probability from $\theta_{\text{P-Vit(G)}}$ and correspond to the last term in the End case of Equation 7.2. In other words, some constituent (at the frontier level, and copied down to D) is completed at every time step. A CER will always result in an AWT at the level above it, and an IER will always result in an ACT at the same level. In time step t+1, the recurring probability will account for the observation at t.

From the final equality, we can see that updating the previous state's information with θ_{R} and θ_{S} will give a probability estimate at each time step, calculated from the level-specific dependencies with estimated attachments. Thus, to finish defining the

[1] Technically, this is not the same IC(G),d as in Equations 7.4 and 7.5, since the max operation is not taken within IC(G),d itself, but in the HHMM transition equations. Similarly, the Vit(G) probabilities are not the same Viterbi analyses from bottom-up parsing, but Viterbi paths through the HHMM trellis. However, it can be shown that the maximum which is found in either placement of the max operator is the same.

HHMM parser, we need to define $\theta_{\text{G-CEE},d}, \theta_{\text{G-AWT},d}, \theta_{\text{G-ACT},d}, \theta_{\text{G-CER},d}$, and $\theta_{\text{G-IER},d}$, which instantiate θ_{R} and θ_{S}.

First, cross-element expansions (CEE):

$$\mathsf{P}_{\theta_{\text{G-CEE},d}}(\langle c_{\eta\iota}, c'_{\eta\iota}\rangle \,|\, \langle -, c_\eta\rangle) \stackrel{\text{def}}{=} \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{*}{\rightarrow} c_{\eta\iota}\ \ldots) \cdot [\![x_{\eta\iota} = c'_{\eta\iota} = c_{\eta\iota}]\!] \tag{7.13}$$

For awaited transitions (AWT), from Equation 7.4:

$$\begin{aligned}\mathsf{P}_{\theta_{\text{G-AWT},d}}(\langle c_\eta, c_{\eta\iota 1}\rangle \,|\, \langle c'_\eta, c_{\eta\iota}\rangle\ c_{\eta\iota 0}) &\stackrel{\text{def}}{=} [\![c_\eta = c'_\eta]\!] \cdot \frac{\mathsf{P}_{\theta_{\text{G-R},d}}(c_{\eta\iota} \rightarrow c_{\eta\iota 0}\ c_{\eta\iota 1})}{\sum_{c_{\eta\iota 1}} \mathsf{P}_{\theta_{\text{G-R},d}}(c_{\eta\iota} \rightarrow c_{\eta\iota 0}\ c_{\eta\iota 1})} \\ &= [\![c_\eta = c'_\eta]\!] \cdot \frac{\mathsf{P}_{\theta_{\text{G-R},d}}(c_{\eta\iota} \rightarrow c_{\eta\iota 0}\ c_{\eta\iota 1})}{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_{\eta\iota} \overset{0}{\rightarrow} c_{\eta\iota 0}\ \ldots)}\end{aligned} \tag{7.14}$$

For active transitions (ACT), from Equations 7.2 and 7.5:

$$\begin{aligned}\mathsf{P}_{\theta_{\text{G-ACT},d}}(\langle c_{\eta\iota}, c_{\eta\iota 1}\rangle \,|\, \langle -, c_\eta\rangle\ c_{\eta\iota 0}) &\stackrel{\text{def}}{=} \frac{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{*}{\rightarrow} c_{\eta\iota}\ \ldots) \cdot \mathsf{P}_{\theta_{\text{G-L},d}}(c_{\eta\iota} \rightarrow c_{\eta\iota 0}\, c_{\eta\iota 1})}{\sum\limits_{\substack{c_{\eta\iota} \\ c_{\eta\iota 1}}} \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{*}{\rightarrow} c_{\eta\iota}\ \ldots) \cdot \mathsf{P}_{\theta_{\text{G}}}(c_{\eta\iota} \rightarrow c_{\eta\iota 0}\ c_{\eta\iota 1})} \\ &= \frac{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{*}{\rightarrow} c_{\eta\iota}\ \ldots) \cdot \mathsf{P}_{\theta_{\text{G-L},d}}(c_{\eta\iota} \rightarrow c_{\eta\iota 0}\ c_{\eta\iota 1})}{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{+}{\rightarrow} c_{\eta\iota 0}\ \ldots)}\end{aligned} \tag{7.15}$$

For cross-element reductions (CER):

$$\mathsf{P}_{\theta_{\text{G-CER},d}}(c_{\eta\iota}, 1 \,|\, \langle -, c_\eta\rangle\ \langle c'_{\eta\iota}, -\rangle) \stackrel{\text{def}}{=} [\![c_{\eta\iota} = c'_{\eta\iota}]\!] \cdot \frac{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{0}{\rightarrow} c_{\eta\iota}\ \ldots)}{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{*}{\rightarrow} c_{\eta\iota}\ \ldots)} \tag{7.16}$$

For in-element reductions (IER):

$$\mathsf{P}_{\theta_{\text{G-CER},d}}(c_{\eta\iota}, 0 \,|\, \langle -, c_\eta\rangle\ \langle c'_{\eta\iota}, -\rangle) \stackrel{\text{def}}{=} [\![c_{\eta\iota} = c'_{\eta\iota}]\!] \cdot \frac{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{+}{\rightarrow} c_{\eta\iota}\ \ldots)}{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta \overset{*}{\rightarrow} c_{\eta\iota}\ \ldots)} \tag{7.17}$$

Using Equations 7.13 through 7.17, it can be shown that this HHMM produces the same probabilities as the original (rightward-depth-bounded) PCFG. First, any cross-element reduction to $f_{r_t^d} = 1$ must be followed by an awaited transition. This means that in the product of probability terms for a valid parse, every '$\overset{0}{\rightarrow}$' expected count term in the numerator will be cancelled by an equivalent term in the denominator. Similarly, every in-element reduction to $f_{r_t^d} = 0$ must be followed by an active transition. This means

		Tree-based syntax	Model-based syntax	Model-based syn+sem
Parsing probs.	$\theta_{\text{Vit(G)},d}$, $\theta_{\text{IC(G)},d}$	7.2–7.5		7.18–7.21
HHMM variables	s_t^d, r_t^d	6.18–6.19		7.29–7.30
Est. left children	$\theta_{\text{G-RL}^*,d}$	–	7.6–7.9	7.25–7.28
HHMM stack eqns.	$\theta_{\text{G-CEE},d}$, etc.	6.23–6.27	7.13–7.17	7.31–7.35

Table 7.1: Equation numbers for related incremental parsing equations.

that in the product of probability terms for a valid parse, every '$\overset{+}{\rightarrow}$' expected count term in the numerator will be cancelled by an equivalent term in the denominator. Finally, assuming any valid parse ends at time T with $f_{r_T^1} = 1$, it must be the case that every cross-element expansion is eventually followed by a cross-element reduction, with an equal number of active transitions and in-element reductions intervening. This means that in the product of probability terms for a valid parse, every '$\overset{*}{\rightarrow}$' expected count term in the numerator will be cancelled by an equivalent term in the denominator. The remaining terms will be ordinary PCFG expansions.

With all expected count terms canceling out in the foregoing analysis, one may wonder whether they can be left out. Indeed, if we are only concerned with the final tree, then the correct cross-level attachments are available at the end of a sentence as in Equation 7.4. However, all intermediate steps will be ranked in an implausible manner and the trees at the top of the beam would be completely unintuitive. Additionally, HHMM implementation needs a beam search to be practical. Thus, expected counts allow for an estimated intermediary ranking.

7.2 Mapping Structured Vectorial Semantics to HHMM Parsing

As HHMM probabilities were defined directly on syntactic CFG rules above, they will now be defined on Structured Vectorial Semantics for CFG rules. Since it took several steps to get to a model-based right-corner HHMM definition, Table 7.1 gives the correspondence between various series of equations.

First, the incremental parsing probabilities need to be updated with referents.

These are derived from the direction/depth-specific right-corner decomposition in Equations 7.2 through 7.5, extended to include semantic referents and relations:

$$\mathsf{P}_{\theta_{\text{Vit(G)},d}}(x_\eta \,|\, lce_\eta) = \max_{\iota,\ \substack{lc_{\eta\iota} \\ x_{\eta\iota}}} \sum_{e_{\eta\iota}} \mathsf{P}_{\theta_{\text{IC(G)},d}}(x_\eta{-}x_{\eta\iota},\ lce_{\eta\iota} \,|\, lce_\eta) \cdot \mathsf{P}_{\theta_{\text{P-Vit(G)}}}(x_{\eta\iota} \,|\, lce_{\eta\iota}) \quad (7.18)$$

$$\begin{aligned}
&\mathsf{P}_{\theta_{\text{IC(G)},d}}(x_\eta{-}x_{\eta\iota 1},\ lce_{\eta\iota 1} \,|\, lce_\eta) \\
&\quad = \max_{lc_{\eta\iota}} \sum_{e_{\eta\iota}} \mathsf{P}_{\theta_{\text{IC(G)},d}}(x_\eta{-}x_{\eta\iota},\ lce_{\eta\iota} \,|\, lce_\eta) \cdot \mathsf{P}_{\theta_{\text{IC(G)},d}}(x_{\eta\iota}{-}x_{\eta\iota 1},\ lce_{\eta\iota 1} \,|\, lce_{\eta\iota}) \qquad (7.19) \\
&\quad = \max_{lc_{\eta\iota}} \sum_{e_{\eta\iota}} \mathsf{P}_{\theta_{\text{IC(G)},d}}(x_\eta{-}x_{\eta\iota},\ lce_{\eta\iota} \,|\, lce_\eta) \cdot \max_{\substack{lc_{\eta\iota 0} \\ x_{\eta\iota 0}}} \mathsf{P}_{\theta_{\text{M-R},d}}(lce_{\eta\iota} \rightarrow lc_{\eta\iota 0}\, lc_{\eta\iota 1}) \\
&\qquad\qquad \cdot \sum_{e_{\eta\iota 0}} \mathsf{P}_{\theta_{\text{L}}}(e_{\eta\iota 0} \,|\, e_{\eta\iota}; l_{\eta\iota 0}) \cdot \mathsf{P}_{\theta_{\text{Vit(G)},d}}(x_{\eta\iota 0} \,|\, lce_{\eta\iota 0}) \cdot \mathsf{P}_{\theta_{\text{L}}}(e_{\eta\iota 1} \,|\, e_{\eta\iota}; l_{\eta\iota 1}) \qquad (7.20)
\end{aligned}$$

$$\begin{aligned}
\mathsf{P}_{\theta_{\text{IC(G)},d}}(x_\eta{-}x_{\eta 1},\ lce_{\eta 1} \,|\, lce_\eta) &= \max_{\substack{lc_{\eta 0} \\ x_{\eta 0}}} \mathsf{P}_{\theta_{\text{M-L},d}}(lce_\eta \rightarrow lc_{\eta 0}\, lc_{\eta 1}) \\
&\quad \cdot \sum_{e_{\eta 0}} \mathsf{P}_{\theta_{\text{L}}}(e_{\eta 0} \,|\, e_\eta; l_{\eta 0}) \cdot \mathsf{P}_{\theta_{\text{Vit(G)},d}}(x_{\eta 0} \,|\, lce_{\eta 0}) \cdot \mathsf{P}_{\theta_{\text{L}}}(e_{\eta 1} \,|\, e_\eta; l_{\eta 1}) \quad (7.21)
\end{aligned}$$

It should be noted that of the extra variables, relations l are maximized together with syntactic categories c, but referents e are summed. These summations over referents should be reminiscent of matrix multiplication in SVS (Chapter 3).

Accordingly, the next step is to encapsulate these distributions in vectors, similar to the vectorization process of Section 3.2. For the θ_{M}, θ_{L}, and $\theta_{\text{Vit(G)},d}$ terms, this is relatively straightforward substitution of matrices and vectors. But a representation is needed for incremental probabilities in $\theta_{\text{IC(G)},d}$, as well; since these probabilities model one referent and are conditioned on another referent, they are matrices $\mathbf{E}$ (compare referential dependencies in θ_{L}, which also produces matrices).

Equations 7.18, 7.20, and 7.21 for the End, Middle, and Begin cases become[2]:

$$\mathsf{P}_{\theta_{\text{Vit(G)},d}}(x_\eta \,|\, lc\mathbf{e}_\eta) = [\![\mathbf{e}_\eta = \mathbf{E}_{\eta\times\eta\iota} \cdot \mathbf{e}_{\eta\iota}]\!] \quad (7.22)$$

[2] Again, these are not the same $\theta_{\text{Vit(G)},d}$ and $\theta_{\text{IC(G)},d}$ because 'max' operations have been pushed to scope over the whole equations, to be handled by a parsing algorithm. This is evident in the fact that there are additional conditioned variables. With the 'max' operations absent, their variables are instead bound by the conditions. These variables have been written in tuples as a foreshadowing of the HHMM equations.

$$\mathsf{P}_{\theta_{\mathrm{IC(G)},d}}(x_\eta{-}x_{\eta\iota 1},\, \langle lc_\eta, lc_{\eta\iota 1}, \mathbf{E}_{\eta\times\eta\iota 1}\rangle \,|\, \langle lc_\eta, lc_{\eta\iota}, \mathbf{E}_{\eta\times\eta\iota}\rangle\ lc_{\eta\iota 0}\mathbf{e}_{\eta\iota 0}) \;=$$
$$[\![\ \mathbf{E}_{\eta\times\eta\iota 1} = \mathbf{E}_{\eta\times\eta\iota}\cdot\mathbf{M}(lc_{\eta\iota}\rightarrow lc_{\eta\iota 0}\ lc_{\eta\iota 1})\cdot \mathrm{d}(\mathbf{L}_{\eta\iota\times\eta\iota 0}(l_{\eta\iota 0})\cdot\mathbf{e}_{\eta\iota 0})\cdot\mathbf{L}_{\eta\iota\times\eta\iota 1}(l_{\eta\iota 1})\ ]\!] \quad (7.23)$$

$$\mathsf{P}_{\theta_{\mathrm{IC(G)},d}}(x_\eta{-}x_{\eta 1},\, \langle lc_\eta, lc_{\eta 1}, \mathbf{E}_{\eta\times\eta 1}\rangle \,|\, \langle -, lc_\eta, -\rangle\ lc_{\eta 0}\mathbf{e}_{\eta 0}) \;=$$
$$[\![\ \mathbf{E}_{\eta\times\eta\iota 1} = \mathbf{M}(lc_\eta\rightarrow lc_{\eta 0}\ lc_{\eta 1})\cdot \mathrm{d}(\mathbf{L}_{\eta\times\eta 0}(l_{\eta 0})\cdot\mathbf{e}_{\eta 0})\cdot\mathbf{L}_{\eta\times\eta 1}(l_{\eta 1})\ ]\!] \quad (7.24)$$

It should be evident that the original SVS composition equation (Equation 3.19 on page 26) is split up, but accurately calculated, among the three cases. The Middle case is essentially an awaited transition, and the Begin case is an active transition (though attached versions will be defined in an HHMM later).

Expected counts in approximated denotations are estimated using value iteration, analogous to Equations 7.6 through 7.9:

$$\mathbf{L}^0(lc_\eta, lc_{\eta 0}) = \sum_{lc_{\eta 1}} \mathbf{M}_{\mathrm{R},d}(lc_\eta\rightarrow lc_{\eta 0}\ lc_{\eta 1})\cdot\mathbf{L}_{\eta\times\eta\iota 0}(l_{\eta\iota 0}) \quad (7.25)$$

$$\mathbf{L}^k(lc_\eta, lc_{\eta 0^k 0}) = \mathbf{L}^{k-1}(lc_\eta, lc_{\eta 0^k})\cdot\mathbf{M}_{\mathrm{L},d}(lc_{\eta 0^k}\rightarrow lc_{\eta 0^k 0}\ lc_{\eta 0^k 1})\cdot\mathbf{L}_{\eta 0^k\times\eta\iota 0}(l_{\eta\iota 0}) \quad (7.26)$$

$$\mathbf{L}^*(lc_\eta, lc_{\eta\iota}) = \sum_{k=0}^{\infty}\mathbf{L}^k(lc_\eta, lc_{\eta\iota}) \approx \sum_{k=0}^{K}\mathbf{L}^k(lc_\eta, lc_{\eta\iota}) \quad (7.27)$$

$$\mathbf{L}^+(lc_\eta, lc_{\eta\iota}) = \mathbf{L}^*(lc_\eta, lc_{\eta\iota}) - \mathbf{L}^0(lc_\eta, lc_{\eta\iota}) \quad (7.28)$$

The matrices $\mathbf{L}^0, \mathbf{L}^k, \mathbf{L}^*$, and $\mathbf{L}^+$ as used here take the place of the $\theta_{\text{G-RL}^*,d}$ model, extended for semantics. In the CEE case, for example, an estimate for an unbounded number of left-child expansions (after a right-expansion) will need to not only estimate the syntactic attachment, but also the corresponding semantic relation. Thus, $\mathbf{L}^*$ will be used to encompass both of these functions. Each approximate syntactic attachment has its own matrix of likely referential correspondents, so the matrices above take syntactic constituents as arguments.

It is also important to emphasize that these approximated matrices carry *expected counts*, not probabilities. Thus, they are directly comparable to the expected counts of Equations 7.6–7.9, including the expected counts of semantic relations and concepts. In incremental processing, how many left-children are implied by a CEE or an ACT is indeterminate.

Now, HHMM modeled states and reduce states can be updated to reflect vectorial semantics. Store elements contain the syntactic and semantic variables for approximately-attached incomplete constituents. Since approximate attachments situate the variables in the context of the depth level above them, extra indices will be needed to describe both levels. Here, η is an arbitrary string, $\iota = 1^i, \kappa = 0^k$, and $\lambda = 1^\ell$.

$$s^d_{\eta\iota\kappa \times \eta\iota\kappa\lambda} \stackrel{\text{def}}{=} \langle\ lc_{\eta\iota\kappa},\ lc_{\eta\iota\kappa\lambda},\ \mathbf{L}^*(lc_{\eta\iota\kappa\lambda}, lc_{\eta\iota\kappa}) \cdot \mathbf{E}_{\eta\iota\kappa \times \eta\iota\kappa\lambda}\ \rangle \tag{7.29}$$

The difference between levels is exemplified by the alternating right and left branches in $\iota\kappa\lambda$ — these are clearly a case of center-embedding. Essentially, this connects the awaited constituents at $d-1$ to the local incomplete constituents being calculated at d.

Reduction states must contain the syntactic and semantic variables at a complete constituent, as well as a flag indicating whether a reduction has occurred. Since reduction states correspond to completed constituents, the semantic vector, $\mathbf{e}$, will contain Viterbi probabilities, just as in non-incremental SVS. These may be preterminal probabilities or other non-terminal probabilities constructed from subtrees:

$$r^d_\eta \stackrel{\text{def}}{=} \langle\ f_\eta,\ lc_\eta,\ \mathbf{e}_\eta\ \rangle \tag{7.30}$$

Embedded in these HHMM variables are chains of matrices and vectors that represent incremental syntactic and semantic information.

HHMM probabilities with vectorial semantics are then derived from the analogous syntax-only versions in Equations 7.13 through 7.17. Syntactic probabilities $\theta_{\text{M-L},d}$ and $\theta_{\text{M-R},d}$ are substituted with matrices $\mathbf{M}_{\text{L},d}$ and $\mathbf{M}_{\text{R},d}$ according to Equation 3.13 and semantic relation probabilities θ_{L} are substituted with matrices $\mathbf{L}$ according to Equation 3.14. Similar to the syntactic version, estimated normalization terms (here, inverted matrices instead of denominators) will cancel out, yielding the same probabilities as the semantics-augmented PCFG.

The complete set of vectorial semantic HHMM probabilities can now be defined. Full derivations are in Appendix A.1, using the assumption $\mathbf{n} \stackrel{\text{def}}{=} \mathbf{1}$ to arrive at these equations.

For cross-element expansions (CEE), set $\eta = *1$ and $\iota = 0^k$. Estimates of the syntactic category and relation are incorporated via $\mathbf{E}_{\eta\iota \times \eta\iota}$.

Incremental SVS Cross-Element Expansion

$$\mathsf{P}_{\theta_{\text{G-CEE},d}}(\langle lc_{\eta\iota}, lc'_{\eta\iota}, \mathbf{E}_{\eta\iota\times\eta\iota}\rangle \,|\, \langle -, lc_{\eta}, -\rangle) = \\ [\![x_{\eta\iota} = lc'_{\eta\iota} = lc_{\eta\iota}]\!] \cdot [\![\mathbf{E}_{\eta\iota\times\eta\iota} = \mathbf{L}^{*}_{\eta\times\eta\iota}(lc_{\eta}, lc_{\eta\iota}) \cdot \mathbf{I}_{\eta\iota\times\eta\iota}]\!] \tag{7.31}$$

Awaited transitions (AWT) are defined with $\eta = *10$ and $\iota = 1^k$.

Incremental SVS Awaited Transition

$$\mathsf{P}_{\theta_{\text{G-AWT},d}}(\langle lc_{\eta}, lc_{\eta\iota 1}, \mathbf{E}_{\eta\times\eta\iota 1}\rangle \,|\, \langle lc'_{\eta}, lc_{\eta\iota}, \mathbf{E}_{\eta\times\eta\iota}\rangle \; lc_{\eta\iota 0}\mathbf{e}_{\eta\iota 0}) \\ = [\![lc_{\eta} = lc'_{\eta}]\!] \cdot [\![\mathbf{E}_{\eta\times\eta\iota 1} = \mathbf{E}_{\eta\times\eta\iota} \cdot \mathbf{M}_{\text{R},d}(lc_{\eta\iota} \rightarrow lc_{\eta\iota 0} \; lc_{\eta\iota 1}) \\ \cdot \mathrm{d}(\mathbf{L}_{\eta\iota\times\eta\iota 0}(l_{\eta\iota 0}) \cdot \mathbf{L}^{0}_{\eta\iota\times\eta\iota 0}(lc_{\eta\iota}, lc_{\eta\iota 0})^{-1} \cdot \mathbf{e}_{\eta\iota 0}) \cdot \mathbf{L}_{\eta\iota\times 1}(l_1)]\!] \tag{7.32}$$

The matrix subscripts on the conditioned state $\mathbf{E}_{\eta\times\eta\iota}$ (previous time step) make it clear that AWT is a incrementally computing the probability of another right-expansion $\mathbf{E}_{\eta\times\eta\iota 1}$.

Notice that this is different from Equation 7.23, which was also said to correspond to the AWT case when vectorizing the $\theta_{\text{IC(G)}}$ equations directly; the inverse matrix $\mathbf{L}^{0}_{\eta\iota\times\eta\iota 0}(lc_{\eta\iota}, lc_{\eta\iota 0})^{-1}$ has been included to cancel out the effects of approximation in $\mathbf{e}_{\eta\iota 0}$, which was a left child and therefore included some approximation over left-expansions.

For active transitions (ACT), we will use $\eta = *1$ and $\iota = 0^k$, where the address $\eta\iota$ is assumed to have children at $\eta\iota 0$ and $\eta\iota 1$:

Incremental SVS Active Transition

$$\mathsf{P}_{\theta_{\text{G-ACT},d}}(\langle lc_{\eta\iota}, lc_{\eta\iota 1}, \mathbf{E}_{\eta\iota\times\eta\iota 1}\rangle \,|\, \langle -, lc_{\eta}, -\rangle \; lc_{\eta\iota 0}\mathbf{e}_{\eta\iota 0}) \\ = [\![\mathbf{E}_{\eta\iota\times\eta\iota 1} = \mathbf{L}^{*}_{\eta\times\eta\iota}(lc_{\eta}, lc_{\eta\iota}) \cdot \mathbf{M}_{\text{L},d}(lc_{\eta\iota} \rightarrow lc_{\eta\iota 0} \; lc_{\eta\iota 1}) \\ \cdot \mathrm{d}(\mathbf{L}_{\eta\iota\times\eta\iota 0}(l_{\eta\iota 0}) \cdot \mathbf{L}^{+}_{\eta\times\eta\iota 0}(lc_{\eta}, lc_{\eta\iota 0})^{-1} \cdot \mathbf{e}_{\eta\iota 0}) \cdot \mathbf{L}_{\eta\iota\times\eta\iota 1}(l_{\eta\iota 1})]\!] \tag{7.33}$$

Similar to the cross-element expansion case, this in-element expansion does not know a priori how far down a left-branching chain it is, so the same estimated matrix $\mathbf{L}^{*}_{\eta\times\eta\iota}(lc_{\eta}, lc_{\eta\iota})$ is included at the beginning of the equation. The syntactic matrix and following operators then take that left-expanding estimate and consider its children.

Only these two cases (AWT and ACT) have grammar rule expansions directly from the direction/depth-specific PCFG (in matrices $\mathbf{M}$). All other expected-count estimations must be normalized via the inverted matrices.

Cross-element reductions (CER) will use $\eta = *1$ and $\iota = 0^k$; also, $\kappa = 1^\ell$ for some constant ℓ, and the address $\eta\iota\kappa$ must correspond to a preterminal:

Incremental SVS Cross-Element Reduction

$$\begin{aligned}&\mathsf{P}_{\theta_{\text{G-CER},d}}(lc_{\eta\iota}\mathbf{e}_{\eta\iota}, 1 \,|\, \langle -, lc_\eta, - \rangle \; \langle lc'_{\eta\iota}, lc_{\eta\iota\kappa}, \mathbf{E}_{\eta\iota\times\eta\iota\kappa} \rangle) \\ &\quad = [\![\mathbf{e}_{\eta\iota} = \mathbf{L}^0_{\eta\times\eta\iota}(lc_\eta, lc_{\eta\iota}) \cdot \mathbf{L}^*_{\eta\times\eta\iota}(lc_\eta, lc_{\eta\iota})^{-1} \cdot \mathbf{E}_{\eta\iota\times\eta\iota\kappa} \cdot \mathbf{n}_{\eta\iota}]\!] \end{aligned} \tag{7.34}$$

Cross-element reductions finish off HHMM stack levels — the matrix $\mathbf{E}_{\eta\iota\times\eta\iota\kappa}$ contains all level-specific information up to the current point, and the original estimate $\mathbf{L}^*_{\eta\times\eta\iota}(lc_\eta, lc_{\eta\iota})$ from the original expansion to this level is nullified by its inverse. In its place, a single-step estimation (corresponding to $\mathbf{L}^0_{\eta\times\eta\iota}(lc_\eta, lc_{\eta\iota})$) connects the higher level of the HHMM to the finishing one.

In-element reduction (IER) are defined with $\eta = *1$, $\iota = 0^k$, and κ maintaining the same restriction as the cross-element case:

Incremental SVS In-Element Reduction

$$\begin{aligned}&\mathsf{P}_{\theta_{\text{G-CER},d}}(lc_{\eta\iota}\mathbf{e}_{\eta\iota}, 0 \,|\, \langle -, lc_\eta, - \rangle \; \langle lc'_{\eta\iota}, lc_{\eta\iota\kappa}, \mathbf{E}_{\eta\iota\times\eta\iota\kappa} \rangle) \\ &\quad = [\![\mathbf{e}_{\eta\iota} = \mathbf{L}^+_{\eta\times\eta\iota}(lc_\eta, lc_{\eta\iota}) \cdot \mathbf{L}^*_{\eta\times\eta\iota}(lc_\eta, lc_{\eta\iota})^{-1} \cdot \mathbf{E}_{\eta\iota\times\eta\iota\kappa} \cdot \mathbf{n}_{\eta\iota}]\!] \end{aligned} \tag{7.35}$$

This operation is quite similar to the cross-element version, except that the finishing constituent is in some string of left-expansions and not at the end of a stack level. Thus, the estimation $\mathbf{L}^+_{\eta\times\eta\iota}(lc_\eta, lc_{\eta\iota})$ leaves out the possibility that it could directly connect to upper stack levels.

7.3 Discussion

The previous section incrementalizes SVS using the right-corner transform to minimize working memory demands during recognition. In deriving equations, though, the assumption $\mathbf{n} \stackrel{\text{def}}{=} \mathbf{1}$ was made. Logical-interpretation SVS in Chapter 5, however, presents

a case in which $\mathbf{n} \overset{\text{def}}{\neq} \mathbf{1}$ — non-existential quantification over a model-theoretic set. Therefore, we must answer the question: what happens if $\mathbf{n}$ cannot be ignored?

Compare a complete constituent vector

$$\mathbf{e}_{\eta\iota} = \mathbf{M}(lc_{\eta\iota} \rightarrow lc_{\eta\iota 0}\ lc_{\eta\iota 1}) \cdot \mathrm{d}(\mathbf{L}_{\eta\iota\times\eta\iota 0}(l_{\eta\iota 0}) \cdot \mathbf{e}_{\eta\iota 0}) \cdot \mathrm{d}(\mathbf{L}_{\eta\iota\times\eta\iota 1}(l_{\eta\iota 1}) \cdot \mathbf{e}_{\eta\iota 1}) \cdot \mathbf{n}_{\eta\iota} \qquad (3.22')$$

with an incomplete constituent matrix from the model-based right-corner transform on SVS:

$$\mathbf{E}_{\eta\times\eta\iota 1} = \mathbf{E}_{\eta\times\eta\iota} \cdot \mathbf{M}(lc_{\eta\iota} \rightarrow lc_{\eta\iota 0}\ lc_{\eta\iota 1}) \cdot \mathrm{d}(\mathbf{L}_{\eta\iota\times\eta\iota 0}(l_{\eta\iota 0}) \cdot \mathbf{e}_{\eta\iota 0}) \cdot \mathbf{L}_{\eta\iota\times\eta\iota 1}(l_{\eta\iota 1}) \qquad (7.23')$$

which is approximated in the AWT case of Equation 7.32. Of course, the right-tail recursion of Equation 7.23′ lacks the complete constituent $\mathbf{e}_{\eta\iota 1}$ (this is what makes the incomplete constituent incomplete); however, Equation 7.23′ also lacks $\mathbf{n}_{\eta\iota}$.

If $\mathbf{n}=\mathbf{1}$ then $\mathrm{d}(\mathbf{L}\cdot\mathbf{e})\cdot\mathbf{n} = \mathbf{L}\cdot\mathbf{e}$ and whenever the awaited $\mathbf{e}_{\eta\iota 1}$ is fully recognized, $\mathbf{n}=\mathbf{1}$ is tacitly applied. But if $\mathbf{n} \overset{\text{def}}{\neq} \mathbf{1}$ then it cannot be tacitly applied for successive awaited transitions; instead, it will be mistakenly (from the perspective of non-incremental SVS) ignored. Thus, some other strategy would have to be used to deal with $\mathbf{n}$ appearing in right-branching structure.

This should not be a surprise. The right-corner transform forced us to estimate successive left-branching children because not all the information is available incrementally. Here, it causes us to make other special arrangements for successive right-branching children.

7.3.1 Predictions

Accordingly, the incremental SVS model makes interesting predictions about memory demands of non-existential quantifiers.

Since the assumption $\mathbf{n}=\mathbf{1}$ does not hold in the case of universal quantification as defined in Section 5, it would therefore be ignored when invoking an awaited transition. In fact, any non-existential quantifier would require a similar non-linear operator $\mathbf{n}$ to carry out the desired cardinality check. Since any $\mathbf{n} \neq \mathbf{1}$ cannot be tacitly applied along with a completed tail $\mathbf{e}_{\eta\iota 1}$, awaited transitions (which expand without invoking a new store element) are not licensed for non-existential quantifiers.

Determiner	Right-Corner Stack Depth							
	1		2		3		4	
all	446	39.7%	577	51.3%	99	8.8%	2	0.1%
both	277	61.0%	165	36.3%	12	2.6%	0	0.0%
each	187	45.3%	199	48.2%	27	6.5%	0	0.0%
every	58	30.4%	112	58.6%	19	9.9%	2	1.0%
half	18	18.6%	59	60.8%	19	19.6%	1	1.0%
no	179	25.9%	445	64.3%	66	9.5%	2	0.3%
Non-exist.	1165	**39.2%**	1557	**52.4%**	242	**8.1%**	7	**0.2%**
Other DTs	29398	**35.2%**	45986	**55.1%**	7901	**9.5%**	248	**0.3%**
All DTs	30563	35.3%	47543	55.0%	8143	9.4%	255	0.3%

Table 7.2: Non-existential quantifiers and other determiners occurring at each stack depth.

With this complexity in non-existential quantification, a correct application of this kind of quantification requires another HHMM store element. The extra store element would then make comprehension more difficult, and that kind of quantified construction would retreat to easier sentences. Thus, we expect to see *non-existential quantification occurring more commonly at lower HHMM depths* than other, grammatically similar words.

This prediction is formally based on the mapping of model-theoretic logic to memory elements in the incremental-SVS HHMM parser. It also accords with intuition, since sentences with nested quantifiers may easily seem difficult to process ('did a person drive a car into a lake?' vs. 'did every person drive no car into two lakes?').

7.3.2 Corpus Analysis

To test this prediction, occurrences of non-existential quantifiers *all, both, each, every, half,* and *no* were examined in the Penn Treebank Wall Street Journal corpus, sections 2 through 21. These were chosen because they were the most frequent non-existential quantifiers tagged as determiners in the corpus. The trees in this corpus were converted to right-corner form, then analyzed to determine the depth of each determiner when mapped to a bounded store of incomplete constituents.

Table 7.2 shows the distribution of these determiners across depths. For each determiner, the frequency count of each determiner is given alongside the percent of time it appeared at a particular stack depth, and percentages in each row add to 100%. The third-to-last and second-to-last rows split the set of determiners into non-existential

quantifiers vs. any other determiner.

Comparing the depth-distributions of these two sets of determiners, it is evident that non-existential quantifiers behave differently than other determiners, in that they are more likely to appear in lower store depths. The difference in percentage is highly significant: $p = 3.5 * 10^{-6}$ on a t-test comparing non-existential quantifiers with all determiners. This suggests that non-existential quantifiers may indeed require more memory resources than existential quantifiers or non-quantifiers, as predicted by the model.

It should be emphasized that this is a very general prediction, tied to the Gricean maxim of manner (Grice, 1975). Interlocuters will tend away from constructions that are more difficult to understand, so deeply-nested non-existential quantifiers are less frequent. There are, of course, exceptions to the rule: for example, *all* and *every* have very different distributions, and *half* appears to be relatively common at depth 3.

Investigation into the latter showed that some sentences had the word *half* incorrectly tagged as a DT. For example,

> *James E. Ousley, computer products group president, said such arrangements help slash Control Data's computer research and development costs in half by the end of 1990*

In this particular sentence, the word shows up in depth 3, a position that the model would predict to be much less likely than lower depths. This sort of tagging error was difficult to programmatically avoid, and may have accounted for some of the outlying data.

Regardless, Table 7.2 shows that, on average, non-existential quantifiers occur at shallower HHMM depths and are therefore likely to require more memory in human sentence processing.

7.4 Summary

This chapter has developed an incremental language model based on a model-based right-corner transform. It extended Structured Vectorial Semantics and then defined an HHMM parser capable of incremental interpretation.

In this incremental SVS framework, the logical interpretation model of Chapter 5 would predict that, on average, non-existential quantifiers appear at shallower HHMM

depths. A corpus study on the Wall Street Journal showed that the predictions made by the HHMM parser with logical-interpretation SVS are indeed validated in natural language text. This suggests that the overall model is linguistically and cognitively plausible.

Chapter 8

Language Modeling Principles

In the introduction to this thesis, I indicated that this research aims to extend and implement a line of computational models for language understanding that are humanly plausible. Since natural language is human by nature, computational models of human language will always be just that — models. To the degree that they miss out on the information that humans would tap into, I would expect to see some engineering benefit to psycholinguistically plausible approaches. Conversely, as has been noted by many others, the defining of computational language models can itself help further the study of cognition in language.

To both of these ends (engineering benefit and scientific discovery), I will now examine the thesis work of the preceding chapters from the perspective of psycholinguistically plausible language modeling. This chapter attempts to be understandable to those who are interested in the psychological contribution of this work but not the details of the computational model. That being said, it may not be as convincing without a thorough understanding of the foregoing chapters.

I will begin with the need for an interactive semantic component in language understanding models (Section 8.1). Then, if we want some type of computational account for meaning, the first question to ask may be "How do people represent that meaning?" so that our models of language can reflect similar attributes (Section 8.2). A related question is "What is the human process of understanding that meaning?" and how to incorporate that into some natural language understanding framework (Section 8.3).

Of course, there are a lot of overlapping topics and conflicting opinions, but the discussion will focus on what is immediately dealt with in incremental Structured Vectorial Semantics (SVS), or easily extensible from it.

8.1 The Existence of Interactive Semantics

The term "language model" is frequently used in Natural Language Processing as a technical term to refer to an n-gram model. It has also been used for the rules associated with a context-free grammar. These are, indeed, models of language — and generally speaking, any probability distribution representing the generation or recognition of language may be called a "language model."

But there is often a lot of information missing. These models are not trying to claim that semantics and pragmatics and discourse do not exist, though they often leave out explicit representations of these components. They are using the most practical information possible for engineering purposes, not for cognitive modeling purposes.

The need for coherent semantic information that interacts with other linguistic processes is likely obvious to a reader. Consider the following sentence:

> But when Little Red Riding Hood noticed some lovely flowers in the woods,
> she forgot her promise to her mother.

Ericsson and Kintsch (1995), in analyzing the memory locale of different types of information, show that cognitive processes result in a sequence of stable states. These states are likely assembled in *short-term working memory* (ST-WM, or just short-term memory) alongside syntax, but quickly integrated into some representation of discourse in *long-term working memory* (LT-WM, or episodic memory). Retrieval cues access information from LT-WM.

Retention tests show that when subjects are interrupted during text comprehension, syntactic structures are forgotten, but meaning may be retained. Thus, it is unlikely that tree structures are the objects that are integrated from ST-WM into LT-WM. Syntactic processing probably occurs in ST-WM, and it is instead a semantic structure that is likely integrated into LT-WM. Without looking, why did Little Red Riding Hood forget her promise in the quote above? It is difficult to recall the exact phrase, but much easier to recall the idea.

Because of this need to store coherent semantic structures from ST-WM to LT-WM, even language models that reach beyond what is traditionally considered "syntactic" are somewhat lacking as *cognitively* plausible language models. For example, Petrov et al. (2006) automatically learn that many of the human-tagged syntactic constituents could in fact be more specific; thus, their grammars are augmented with *latent annotations* that greatly improve their performance on the standard syntactic parsing task. Along with other phrase structure, parts of speech are shown to divide into very interpretable clusters, so that semantically similar words will often get their own latent annotation. Thus, lexical semantic information has been represented by the latent anontations, and has aided in syntactic ambiguity resolution.

(Some may disagree with the possibility of semantics influencing the syntactic ambiguity resolution, relying solely on structural criteria (Bever, 1970). However, more recent work shows that lexical information is in fact used (Ford, Bresnan, and Kaplan, 1982; MacDonald, Pearlmutter, and Seidenberg, 1994), and that other sources of information such as presuppositions are taken into account as well (Crain and Steedman, 1985).)

Latent annotations correctly model the perspective that lexical semantics influences decisions on syntactic ambiguity resolution; but unfortunately, resolving syntactic ambiguities is not the only task that occurs during human language processing. These annotations only have meaning in the context of their syntactic constituents. They cannot be applied to problems outside of syntactic ambiguity resolution, or used as a meaning representation.

Without a model of semantics (or meaning, or intention), then, computationally-modeled linguistic sub-processes (like parsing) may have difficulty where people would not. (Or, they may even *not* have difficulty where people *do*.) This argues in favor of having a semantic representation that may interact with other linguistic processes, rather than considering semantics a sub-task of parsing.

SVS makes semantic interpretation fully interactive with syntactic parsing. Chapter 3 introduced a probabilistic factorization of syntactic–semantic grammar rules into

roughly syntactic and semantic parts:

$$\mathsf{P}_{\theta_{\mathrm{G}}}(lce_{\eta} \rightarrow lce_{\eta 0}\ lce_{\eta 1}) \stackrel{\text{def}}{=} \mathsf{P}_{\theta_{\mathrm{M}}}(lce_{\eta} \rightarrow lc_{\eta 0}\ lc_{\eta 1}) \cdot \mathsf{P}_{\theta_{\mathrm{L}}}(e_{\eta 0} \,|\, e_{\eta}; l_{\eta 0}) \cdot \mathsf{P}_{\theta_{\mathrm{L}}}(e_{\eta 1} \,|\, e_{\eta}; l_{\eta 1}) \qquad (3.1')$$

where θ_{M} is mostly syntactic, and θ_{L} is semantic. This factorization allows a semantic processing model to reside directly alongside a syntactic processing model, and be considered together during recognition. Though other processes such as discourse integration are not explicitly modeled in SVS, they are possible because vectors (encapsulating the probability distributions above) cohesively represent semantic information.

SVS assumes that interactive, cohesive semantic information is necessary for later (unmodeled) discourse processes like coreference resolution. This information is modeled and available in the form of semantic vectors. As future work, then, the framework is extensible to the modeling discourse integration processes.

8.2 Mental Representations of Meaning

It is a fundamentally difficult (perhaps circular) problem to define *meaning*, but the fields of semantics, pragmatics, and discourse all study how we as humans obtain and represent that meaning. I will present an assortment of clues and historic approaches to the study of meaning, drawing connections to SVS (without necessarily assuming the validity of each theory, since many disagree).

Family Resemblance

Wittgenstein (1953) invited readers to define the word "game" in one of his "thought experiments." A definition based on competitiveness is ruled out by the game of catch (tossing a ball back and forth). A definition based on opposing sides is not consistent with role-playing games. Definitions based on mirthful enjoyment, structured rules, etc. must all be similarly discounted.

One way to deal with this is to divide between the senses of a word. WordNet in English (Fellbaum, 1998) does so with vigor, leading to 11 definitions of the noun "game." Some of the assumptions in treating lexical semantics this way are *separability* and *comprehensiveness*: that the lexeme can be studied on its own, and that any

processing of natural language must have the whole store of possible word senses that humans do.

On the opposite side of the spectrum, Wittgenstein's original assertion was that boundaries of meanings are not to be drawn; rather, meaning is simply to be comprehended as enough "family resemblance" is seen. If enough characteristics of an activity sound like other things in the "games" family, that activity is also a game.

Landauer and Dumais (1997) and Kintsch (2001) consider Latent Semantic Analysis to be a computational theory of this sort of distributed 'family resemblance' meaning. This is because Latent Semantic Analysis and related vector-space techniques take all the information of how words are distributed in text and boils them down into the most important concepts; words or phrases may then express (or resemble) these concepts to varying degrees.

Distributed, quantitative approaches like LSA were the original motivation for choosing vectorial representations of meaning in this thesis. Other vector spaces models have extended LSA (Hofmann, 2001; Blei, Ng, and Jordan, 2003), giving a wide range of possible instantiations to SVS. Chapter 4 uses similar techniques with vectorial representations to apply meaning in a "family resemblance" manner; there, a "family" relationship is a head–modifier relationship, and words that are used in similar relationships cluster together into concepts with stronger or weaker probabilities.

Underspecified Semantics

Another consequence of a structured, vector-space definition of semantics is that words with underspecified semantics have a natural representation. Johnson-Laird (1987) proposes two types of underspecification in the mental lexicon: those where semantics have *yet to be acquired*, and those that are *intrinsically incomplete*.

With LSA-style vectors, Kintsch (2001) likens vector magnitude to how much information the model has about a particular word or phrase. In his example, the word "horse" has a vector length of 2.49, "porch" has a length of .59, and "the" has a vector length of 0.03. This indicates that seeing the word "horse" is a powerful constraint; "porch" might not have been used as frequently in the data and could require a bit more context to fill out why it is being used; and very little complete meaning actually resides in the word "the." These three cases loosely correspond to well-specified,

yet-to-be-acquired, and intrinsically-incomplete words.

A similar connection can be applied to the vectors in SVS. Resolving intrinsically incomplete words during parsing is accomplished through the same structured vector composition process as words with more-complete semantics. The idea is that if a word with intrinsically incomplete semantics is observed, its relation to other words will take that into account. This standard process of utilizing context through the vector composition process will also be used to resolve yet-to-be-acquired lexical entries — we will revisit this later.

Words or phrases may also be intrinsically incomplete for a somewhat orthogonal reason, not based on lexical semantics: it may be because their semantic scope or resolution is yet undetermined in an utterance, e.g., for quantifiers. Approaches like Hole Semantics (Bos, 1996) or Quasi-Logical Forms (Alshawi, 1992) explicitly represent unscoped quantifiers and unresolved relations by developing meta-level languages that allow for this kind of of underspecification.

Interestingly, *relations* in the logical-interpretation instantiation of SVS take on quantification and similar cases (see the definition of universal quantification on page 52). Thus, with incremental SVS, such relations may in fact apply while semantic arguments to those relations are still unobserved, modeling unresolved relations and scope ambiguities as part of the regular process of semantic interpretation.

Logic

Because formal semantics is primarily a study of *logic*, many theoretical frameworks integrating semantics and syntax have taken this approach. For example, Quasi-Logical Form and Hole Semantics are both formulated as meta-languages for formal logics. Dynamic Predicate Logic (Groenendijk and Stokhof, 1991) extends regular predicate logic in order to account for some degree of incrementality in interpretation. Zwarts and Winter (2000) incorporate the semantics of spatial relations into a logical framework using vectors (which they also term "vector-space semantics").

Because of this rich tradition, Chapter 5 gave an instantiation of SVS that was defined in terms of first-order logic. Vectors were defined as sets of denoted individuals, and matrices were ordered pairs of individuals, representing some relationship between those individuals. Similar approaches in previous work (Wu, Schwartz, and Schuler, 2008a;

Schuler, Wu, and Schwartz, 2009) have used such integrated, model-theoretic frameworks for spoken language interfaces; the casting of these models into a vectorial notation does not invalidate their underlying representational power, as long as matrix relationships can successfully encode the basic logical operators. Thus, the worked example of first-order logic operations in Chapter 5 indicates that SVS is well able to handle this type of semantics.

In reality, both "family resemblance" semantics and logical interpretation seem to be in play for human language processing. Again drawing on Johnson-Laird's work in lexical semantics (1987), we may consider lexical items to be represented in two distinct ways. First, they are represented in terms of relationships to other words, similar to a semantic network. Second, they are represented in terms of "ineffable truth conditions," or semantic primitives (but these primitives are not directly or measurably tied to lexical items).

In relationally-clustered SVS, vectors computationally represent this first, relational type of information. Most vector-space models will use the "same-document" relationship to produce words with similar topics; Chapter 4 used head–modifier relationships to produce words that are used in similar ways in sentences. Logical-interpretation SVS, on the other hand, could be construed as dealing with the semantic primitives.

The key here is that SVS could very easily include both kinds of semantics. Chapter 5 introduced non-probabilistic matrices and vectors, as well as block-matrix structure, in order to accomplish the first-order predicate logic operations. Extending this block-matrix structure, both relation matrices and semantic vectors could easily include a reduced-dimensionality block and a logical block. Block-diagonal matrices would enforce a clean distinction between these two representations; upper or lower triangular structures would allow distributed semantics to affect logical interpretation, or vice versa.

Models of Discourse Structure

Because of its ability to represent semantic meaning, SVS is amenable to extensions that model discourse structure as well. Hobbs (1979) used predicate logic as a proxy for discussing the idea of *coherence relations* in discourse; he formally defined ways for sections of texts to relate to each other and thus form cohesive wholes. Webber et al.

(2003) instead take the stance that adverbial *cue phrases* work anaphorically in text, rather than operating over whole sections of texts.

The important connection here is that as long as discourse coherence is accomplished by some kind of relation, it can extend the SVS framework. Webber et al. indicate that one of the best strategies for formalizing their model is to extend sentence-level grammars to deal with discourse. Since SVS is defined on PCFGs, this is a natural extension — existing intra-sentential relationships could as easily be augmented with discourse relations as long is there is data from which those relations can be learned. A full-fledged discourse integration component would make use of such relations.

Another discourse phenomenon indicating the need for an additional discourse model is the problem of coreference resolution. Theories of discourse (Grosz and Sidner, 1986) distinguish between *linguistic structure*, *intentional structure*, and *attentional state*. Centering Theory (Grosz, Weinstein, and Joshi, 1995) formalized a framework of the attentional state's influence on coreference resolution.

In a similar vein, the Givenness Hierarchy (Gundel, Hedberg, and Zacharski, 1993) connected linguistic structure and attentional state. Each cognitive state could be realized by particular linguistic structures in a particular language; these cognitive states are ranked (top line, shown with English linguistic structure underneath):

in focus	>	activated	>	familiar	>	uniquely identifiable	>	referential	>	type identifiable
{it}		{this, that, this N}		{that N}		{the N}		{indefinite this N}		{a N}

Given some cognitive state, only cognitive states lower on the hierarchy are allowed to use their linguistic structure.

Although SVS does not itself contain a model of attentional state, it is uncomplicated to conceive of a full-fledged discourse integration model representing operations mapping the semantic representation in ST-WM (that which is calculated by SVS) to LT-WM. Attentional state could be defined from the linguistic context within a probabilistic hypothesis.

Context in Interpretation

With a "family resemblance" account of lexical semantics, words do not completely specify their particular sense without context. Kintsch offers the following example: (Kintsch, 2001)

> *the cat ran away*
> *my horse ran last*
> *a breeze ran through the trees*

In each case, the subject–predicate relationship affects the meaning of the verb "ran." Thus, the meaning of "ran" is dependent on its *context*. Even if the brain stores a lexical entry for "ran," it is unlikely that every sense of the word is considered from the start in parallel; rather, context helps pick out the word sense or the most-salient attribute of that word (Johnson-Laird, 1987). This context-dependence is a type of felicity constraint (Roberts, 2004): does the context make the use of the word reasonable and relevant?

Words and sentences are not just dependent on the context, they also *update* the context. Both the event of an utterance itself and the content of that utterance are part of the update. So the context is not stagnant; meaning is *dynamic*, and it is interpreted incrementally.

To combine the two observations above about context dependence and context updates, meaning is a function that begins with a context and results in another context (Roberts, 2004).

But what is context? This thesis assumes that ambiguity-resolving context can come in any form, e.g., lexical effects, presuppositions, or discourse context. These are all weighed together in the space of mental representations. Then, this wider sense of the "context" is at work in SVS at every level; with mental representations modeled as vectors and matrices, new contexts are computed by updating a beginning context (semantic vector) with some new contextual information (another vector). This process seems to hold both on an intra-sentential scope and on the discourse level.

A more technical use of the word "context" is used in dynamic theories of interpretation (Heim, 1982; Groenendijk and Stokhof, 1991), where it is assumed to be composed of (with some variation):

1. Common ground: propositions held in common by all interlocutors.

2. Discourse referents: real or imagined entities such as participants in a story, the last event in a sequence, or the last paragraph of information.

Also, some way for ordering the relevance of information is necessary. A discourse-level storing of completed intra-sentential constructs would be necessary to extend SVS to be compatible with this definition.

8.3 Manner of Interpretation

The question of how meaning is represented in the mind is integrally tied to the question, "How does it get there?" In this section, I explore several features of an interpretation process that are modeled by incremental SVS, though many of these characteristics are not often considered separately.

Statistical Ranking

Using statistical information in the study of languages has been common for several decades. We may ask why statistical methods have come to dominate the field of Natural Language Processing.

One reason is that it is well-suited for the problems intrinsic to natural language. In developing Relevance Theory, Sperber and Wilson (2004) indicate that greater positive cognitive effects imply that an utterance is more *relevant*; optimally relevant utterances must be the "most" relevant (and worth the processing effort). This type of need for ranking is a common one in linguistic theories, and it is naturally solved through probabilities.

Probabilities are also a good fit, practically speaking. While considering the wide variety of models of *access* and *disambiguation*, Jurafsky (1996) found probability theory to be the only way to integrate multiple models and rank the importance or likelihood of their results, Brants and Crocker (2000) defined a fully probabilistic parser for its psycholinguistic properties. Finding that probabilities benefitted both modeling ease and empirical results, generations of models followed suit, and SVS is one among the ranks of models that have been defined in almost completely probabilistic terms.

A related question, though, is whether probabilities are really cognitively plausible. Johnson-Laird (1994) answers this in the affirmative for the process of language understanding. People construct mental models of what their linguistic stimulus may mean, and find it incumbent upon themselves to estimate the strength of an argument. This inferential strength is defined as the proportion of possibilities compatible with the premises in which the conclusion is true — something that probabilities are well-suited for calculating, and that humans naturally tie to their (sometimes faulty) intuitions about likelihood.

Parallel Hypotheses

Related to this notion that multiple hypotheses can be ranked with probabilities, SVS also naturally represents the concept of *parallel hypotheses*, i.e., that we are simultaneously considering multiple anlyses. Current accounts of sentence processing (Altmann and Steedman, 1988; Hale, 2003; Levy, 2008) assume that alternative analyses of sentence processing are initially offered in parallel. Eye-tracking studies (Tanenhaus et al., 1995; Brown-Schmidt, Campana, and Tanenhaus, 2002; Dahan and Gaskell, 2007) have shown that people focus their attention on plausible sentence completions of spoken language, and that ambiguity in their predictions leads to divided attention between the possible completions.

In past work (Schuler, Wu, and Schwartz, 2009; Schuler et al., 2010, see Chapter 6), parallel *syntactic* hypotheses were represented through the Hierarchical Hidden Markov Model's trellis states. At any given time step, there are multiple states corresponding to possible explanations of the observed data. A most likely sequence that best explains syntactic information is chosen at the end of sentence processing, from continuations of the existing parallel hypotheses.

One may assume that meaning interpretation operates in this parallel manner as well, e.g., in the following sentence:

> *It wasn't until Joan got to the bank that she finally got her feet wet with the basics of financial trading.*

Without prior discourse information, this example shows how context helps choose between completely different word senses for "bank" and different interpretations for

"got her feet wet." Parallel hypotheses in this case would have both senses of "bank" activated (perhaps to different degrees), and the financial-institution account would be chosen when disambiguating information was made available.

This may not account for everything happening in the sentence. Two alternative explanations: later information calls for a process of reconstructing the sentence with each unique word sense; or, the semantics of "bank" are left ambiguous until complemented by the context. It would be strange, however, if a phenomenon in speech processing was completely absent in the processing meaning at some level. In particular, the eye-tracking studies mentioned above show that people keep physical entities in parallel, and these entities would theoretically constitute a grounded, model-theoretic world model. If grounded semantic hypotheses are available in real-time human speech recognition, it would be unlikely for them to be absent in real-time human text comprehension.

An account of parallel hypotheses mean that at any given point in time, there are multiple possible interpretations of a sentence that are possible with some probability. As such, it fits neatly into the HHMM parser framework described above; for incremental SVS, then, trellis states and most likely sequences will produce and rank hypotheses that consider both syntactic and semantic information in parallel.

Incrementality

Multiple hypotheses are typical in probabilistic accounts of language. However, their parallel nature is dependent on *incrementality*, that is, the view that language processing occurs on-the-fly. This account of language processing is incompatible with CKY (or any bottom-up) parsing. The $\mathcal{O}(n^3)$ asymptotic runtime and inability to produce any output until whole sentences are complete disqualifies bottom-up parsers from interpreting continuous input streams. Thus, cohering discourse relations over sections of text (Hobbs, 1979) could be handled, but anaphoric cue phrases (Webber et al., 2003) could not.

In syntatic HHMMs, the right-corner transform of Chapter 6 puts trees in a very intuitive incremental form. Then, (partial) trees in right-corner form are recognized with a Hierarchical HMM, which models the change in possible analyses over time. Because of the right-corner representation, there is a representation for *incomplete constituents* in subtrees whose string yields are not fully observed yet.

It has been suggested (Marslen-Wilson, 1975; Altmann and Steedman, 1988) that it is not just the syntactic interpretation that is incremental, but also the integration of other types of information. In incremental SVS, the model-based right-corner transform and casting to HHMM exemplify this same incrementality with incomplete semantics. This incrementality is crucial in a plausible accounting of dynamic context.

Bounded Short-Term Memory

The work of Ericsson and Kintsch (1995) discussed above implies that syntax operates within the traditional short-term memory store that has been well-studied in psychology. An initial estimate on the size of this short-term memory store was seven elements (plus or minus two) (Miller, 1956). However, later estimates consider only four memory elements for any STM activity (e.g., Cowan (2001)). Regardless of the specific number of elements, this means that some significant part of language must be handled in a short-term memory that is bounded.

Using the HHMM parser for broad-coverage parsing, Schuler et al. (2010) show that an estimated 4 HHMM depth levels (and hence, memory elements) account for virtually all of the possible grammatical constructions in the Wall Street Journal portion of the Penn Treebank. This is because center-embedding is known to increase memory requirements (Miller and Chomsky, 1963; Gibson, 1998), and each HHMM depth level represents a center-embedded structure.

Recognizing semantics together with syntax as in SVS, then, makes the assumption that constructing an initial semantic representation occurs in short-term memory and is thus memory-bounded. This is possible, given the definitions of ST-WM and LT-WM and several hypotheses that an initial semantic structure is constructed before integration into the mental representation of discourse structure (Kintsch, 1988; Johnson-Laird, 1987). However, it also shows that a separate discourse model is the correct, plausible next step, such that a larger memory store may be used for discourse phenomena like coreference.

Linguistic Complexity and Costs

Studies of linguistic complexity give some verbal stimulus and ask the question, "How difficult is it to understand this?" — measured by eye-tracking experiments or reading times. This is useful for teasing apart different cognitive processes, and determining when processes share resources such as memory. Such studies have validated many of the principles listed in this chapter.

Recent studies (Pynte, New, and Kennedy, 2008; Demberg and Keller, 2008; Roark et al., 2009; Boston et al., 2008; Frank, 2009; Wu et al., 2010) typically use the data to evaluate whether a *computational* model of comprehension is a good one. Namely, given the same verbal stimulus, will the model give eye-tracking times or reading times similar to human responses? The method, then, directly evaluates how similar a model is to a human with respect to eye-fixation durations or reading times.

Different metrics of linguistic complexity have been proposed that may be calculated on computational models; *surprisal* (Hale, 2003), in particular, has received widespread attention and use, perhaps due to its information-theoretic properties (Levy, 2008). This has led to many variants which aim to tease out the truly relevant cognitive processes that should be modeled.

Better language models, in principle, lead to better metrics (Frank, 2009). Previous work (Wu et al., 2010) has shown that surprisal calculated with a syntactic HHMM parser is an excellent predictor of human reading time; it also made the link between a filled short-term memory buffer (bounded by HHMM depth level) and linguistic complexity.

Thus, future work may include tests to tease out the contribution of the semantic model to reading times or eye-tracking data, and the validity of syntactic–semantic complexity metrics like surprisal.

8.4 Summary

This chapter has focused on the addition of a flexible and interactive semantic component like SVS as necessary for a full accounting of the semantic, pragmatic, and discourse phenomena exhibited in natural language. Cognitive principles of meaning representation and manner of interpretation undergird the claim that Structured Vectorial

Semantics, especially in incremental form, is a psycholinguistically plausible language model.

Chapter 9

Related Work

This chapter places the language models of this thesis in the context of related work in: vector semantics and clustered representations of meaning (Section 9.1); parsing and related syntactic processing (Section 9.2); semantics combined with syntax (Section 9.3); and semantic interpretation of natural language utterances (Section 9.4).

9.1 Semantic Spaces

Approaches to constructing semantic spaces in information retrieval often begin with the *distributional hypothesis* (Harris, 1954), i.e., that words can be judged 'by the company they keep.' In this line of reasoning, words that occur in the same documents are similar to each other; thus, matrices storing word co-occurrences are high-dimensional semantic spaces representing each word (Lund and Burgess, 1996).

Because the these co-occurrence matrices were sparse, more efficient representations were sought. Latent Semantic Analysis (LSA) used a linear algebra technique, Singular Value Decomposition, for factoring the matrix into a more dense and compact representation (Deerwester et al., 1990). This representation contained the most important information of word co-occurrence matrices, and could be viewed as a projection from the high-dimensional space to the lower-dimensional one. Words with similar co-occurrence patterns in documents were still similar after the dimensionality reduction. LSA was considered to be a plausible cognitive model for meaning representation and reasoning (Landauer and Dumais, 1997; Kintsch, 2001).

LSA was not without shortcomings, however. LSA vectors could only be compared to each other geometrically, and thus the semantic space had no meaning with reference to constructs outside of the space itself. Therefore, Hofmann (1999; 2001) introduced a statistically sound model (the Aspect model) which connected word co-occurrence matrices with words in the framework of probability theory. The resulting approach, probabilistic Latent Semantic Analysis (pLSA), projected words onto the dimensionality-reduced semantic space according to Kullback-Leibler divergence.

Latent Dirichlet Allocation (LDA) extended this to probabilistically model the generation of the documents themselves (Blei, Ng, and Jordan, 2003). As a generative model, LDA chose a distribution of topics for a document, then generated a topic for a word, then generated the word itself. Many other topic models (Griffiths and Steyvers, 2002; Steyvers and Griffiths, 2007; Blei and McAuliffe, 2008) may be considered variants of LDA.

Another distributed-meaning representation, but with different probabilistic assumptions, is Koo and Collins (2005), from the perspective of lexical semantics. They use clustering algorithms to discover senses for polysemous words, and put other words into cohesive clusters. The information presented by the clusters to a downstream task is thus the most relevant information about the lexical items.

An assumption in all of the work above is that of *exchangeability*, so that word order is ignored, as is any notion of syntax. This is also known as the bag-of-words assumption; the only relevant relationships between words in these approaches is which the 'same-document' relationship. Blei, Ng, and Jordan (2003) point out that they also assume the exchangeability of documents within a corpus.

At the document level, Taskar, Segal, and Koller (2001) model the rich dependency structures between documents, found in domains such as scientific papers (overlapping topics, authors, and other features) or the internet (hyperlinks). Information is propagated between documents via the relationships between each document's class variables.

Structured Vectorial Semantics, as defined in Chapter 3 and instantiated in Chapter 4, defines a semantic space similar to pLSA and its descendents. However, it differs crucially in the locus of the exchangeability assumption. Instead of 'same-document' relationships, words are defined as co-occurring in the context of some other relationship, such as the head–modifier dependency relationship. Though techniques from document

relationships may be (and are) used in SVS, document-level dependencies are not explicitly modeled.

Montague (1973) defined semantics to be compositional, and SVS follows in this line by composing vectors to yield full semantic representations of sentences. This composition is done in the context of syntactic–semantic relationships, and intermediate (or resulting) vectors are representations of context.

In Word Sense Disambiguation, co-occurrences have been composed to create dimensionality-reduced descriptions of semantic context. Pedersen and Bruce (1997) use first-order co-occurrences (as described above) to represent context; Schütze (1998) used second-order co-occurrences (words in any document that has a word in common with the word in question) to create clustered contexts.

Kulkarni and Pedersen's (2005) SenseClusters system was developed to solve the related problem of Word sense Discrimination, but may also be applied to headless clustering. A meaning-bearing clustered vector is constructed from lexical features such as unigrams, bigrams, and co-occurrences to represent context. Labels are then automatically added, which characterize the contents of the clusters for interpretability. Reisinger and Mooney (2010) instead cluster words into distinct senses, such that each word has a set of vectors to represent it. Context-dependent meaning is then constructed by combining the nearest vector in the word's vector set to the context vector.

More generally, Mitchell and Lapata (2008) examine the different ways in which vector semantics may be composed. Most information retrieval techniques used the vector/context centroids (essentially additive models); Mitchell and Lapata note that multiplicative (dot product) models actually correlate better with human similarity judgments. In addition, they provide a composition equation for including other syntactic and semantic information: $\mathbf{p} = f(\mathbf{u}, \mathbf{v}, R, K)$ where $\mathbf{p}$ is a resulting composed vector (or tensor), $\mathbf{u}$ and $\mathbf{v}$ are the vectors to be composed, R is syntactic context, and K is a semantic knowledge base.

These approaches are similar to SVS in their representation of semantic context by clustered (WSD) or composed (Mitchell and Lapata) vectors. In fact, Structured Vectorial Semantics may be neatly described within the framework of Mitchell and Lapata's composition equation above, where the function $f(\cdot)$ would be defined by the the SVS composition equation (Equation 3.22) on page 27. However, their evaluated

models do not actually incorporate R, the syntactic context.

9.2 Syntactic Processing

In statistical Natural Language Processing, parsing is the process of choosing the best grammatical structure that explains an observed sentence (see Section 2.2 and following for further background). Initial gains in parsing used lexicalization (see Section 2.5) because of linguistic theories that the verb of a sentence (or more generally, the lexical head of a constituent) determined the rest of the sentence. This improved parsing accuracy on the standard parsing task (Magerman, 1995; Collins, 1996; Collins, 1997). SVS as defined in Chapter 3 can easily represent parsing with bilexical dependencies similar to the probabilistic dependencies in Eisner (1996) and Eisner and Satta (1999).

Later approaches using lexicalization (Charniak, 2000; Charniak and Johnson, 2005) were able to produce even higher accuracy results by using Markovization and coarse-to-fine approaches. These approaches also reranked the n-best parses for the highest accuracy.

However, later analysis has shown that the gains of lexicalization are not primarily from bilexical dependencies (Gildea, 2001; Bikel, 2004). Klein and Manning (Klein and Manning, 2003) introduced linguistically-motivated state-splits that achieved accuracy on par with state-of-the-art parsers at the time. Features such as vertical and horizontal Markovization, special treatment of unary rules, POS tag splitting, head (not headword) annotation, and base NPs markings all contributed to high accuracy parsing.

Following on this intuition that parsing-relevant information was available by state-splitting with other means than lexicalization, Matsuzaki, Miyao, and Tsujii (2005) introduced an automatic state-splitting technique, producing *latent* annotations for each constituent in the grammar. This approach is the most similar to relational-clustering SVS (Chapter 4, with the latent annotations in the language trained via EM). However, SVS factors the 'annotations' as their own separate, semantic component; this enables the annotations to be their own, cohesive representation. Matsuzaki, Miyao, and Tsujii's annotations differ for each grammatical constituent, and thus no semantic representation is available corresponding to the top parse.

Similar techniques (Petrov et al., 2006; Petrov and Klein, 2007; Petrov and Klein, 2008) combine automatic state-splitting with coarse-to-fine techniques to achieve accuracies on par or better than the best lexicalized parsers. The above approaches may automatically learn interpretable state-splits (e.g., the POS splits in Petrov et al. (2006)), even exhibiting properties of semantic similarity within each annotation. Though they bear semantic information, these annotations are only defined locally on individual phrase structure rules and are therefore unable to compose a cohesive semantic representation, as before. Relational-clustering SVS clusters the headwords, thereby including a smoothed version of lexical dependencies that refine syntactic hypotheses as in the state-splitting techniques, but also produce a cohesive semantic representation.

In addition to having language models that are incapable of interactively producing semantic output, most of the above parsers are based on bottom-up chart-parsing algorithms that operate on asymptotic $\mathcal{O}(n^3)$ time. So although they perform nearly as well on the parsing task as humans do (judged by inter-annotator agreement on the Treebank), they are implausible as cognitive models for the human sentence processing mechanism — which is likely incremental and should support linear time inference. Additionally, they do not penalize sentences for containing center-embedded phrase structure (Abney and Johnson, 1991), which is known increase linguistic complexity and reduce recognition time and accuracy.

Roark's (2001) parser is a better model for the psycholinguistic criteria of producing incremental constituents; fully-connected partial derivations specify all syntactic context up to a given word and leave no underspecified representations. However, this model parses in asymptotically polynomial time. Other psycholinguistically-motivated attempts at incremental syntactic processing, such as Cascaded Markov Models, do not penalize for center-embedded structures (Brants, 1999; Brants and Crocker, 2000); these particular approaches also do not report results on the standard Wall Street Journal parsing task.

HHMM parsers (Schuler et al., 2008; Schuler et al., 2010) process input incrementally with linear time inference. Additionally, because dependencies are for right-corner transformed trees (the left–right dual of the left-corner transform (Johnson, 1998)), a penalty is paid for center-embedded structures. This approach to parsing is described

in Chapter 6 as the parsing paradigm for incremental SVS. Thus, incremental SVS represents a psycholinguistically-plausible full parser that produces semantic output.

9.3 Combining Syntax and Semantics

Many previous approaches to combining syntactic context into the types of semantic spaces discussed above have been explored, but only in the context of syntactic dependency structure. Dependency parsers (Nivre et al., 2007; Surdeanu et al., 2008) find the most likely head–modifier hierarchical relationships between words without specifying constituents; this is in contrast to the parsers described above (which are sometimes called constituent parsers). So in dependency parsers there are no nonterminal symbols; but, dependency arcs may optionally be labeled.

Dependency parsing is thus a lightweight, flexible alternative to full parsing. Nivre (2007), for example, presented a deterministic, linear time dependency parser that was also accurate. In fact, SVS may be easily defined on dependency parsers by using constituent-less syntactic models (e.g., removing c from θ_{M} in Equation 3.1)). Because of the flexibility and computational benefits of this model, more semantic information has been integrated into dependency parsers (Surdeanu et al., 2008).

For example, Koo, Carreras, and Collins (2008) use word clusters as features in the discriminative training of a dependency parser. This showed that clustered lexical information has an effect on what is typically considered to be a purely syntactic task, dependency parsing. This is very similar in spirit to relationally-clustered SVS; however, the clusters are not composed to build a representation of meaning during parsing.

Finkel, Grenager, and Manning (2007) use nonparametric Bayesian methods to automatically split POS tags into a conceivably infinite number of states. Their 'infinite tree' applies the same methods used on topic models to learn distinctions on a more local scale, to improve dependency parsing. Thus, despite the similarity in methodology to topic models, the resulting split tags are similar in semantic scope to state-splitting constituent-parsers.

Comparing their work to the infinite tree, Boyd-Graber and Blei (2008) present a variant of a Hierarchical Dirichlet Process that more explicitly models the generation

of dependency structure alongside the generation of semantic topics. Topics are organized in tree dependency structure, and they emit words or other topics. This explicit modeling of the head dependencies decreases the perplexity of the model.

These nonparametric methods give syntactic states that are not constrained to an existing set of tags or constituents; thus, attempting parsing methods in which syntactic models are part of the generative process is difficult. The problem of grammar induction alone is not easily solved by nonparametric methods, so such methods would be hard-pressed to capture both syntactic (constituent) and semantic (topic) information at once in a parser, as SVS does.

Dependency structure has been used outside of dependency parsing, as well. For example, Lin's (1998) automatic thesaurus extraction system used full-fledged parses to find dependency triples, over which semantically similar words were clustered.

Instead of integrating topic models with a dependency structure, Griffiths et al. (2005) use a composite model that models syntactic dependencies as a time-series model, more in line with HMM POS tagging models than dependency parsing. The intuition behind their model is that function words serve only a syntactic purpose and are therefore naturally handled by time-series dependencies, whereas content-bearing words need to be generated by a topic component.

Dependency relations are also used by Padó and Lapata (2007) to produce vector spaces for words that take syntax into account. First, context is built up as some (sub)set of paths in the dependency structure around a target word. Paths are then mapped to basis elements (i.e., vector indices) with some weighting scheme to give corresponding real-number values. The result is similar to SVS in that their vectors bear syntactic and semantic information, just as Viterbi probabilities in SVS are vectors that subsume both syntactic and semantic information. However, the vectors in this context are centered around a target word, and are therefore not being composed in their syntactic and semantic context.

Several other approaches do attempt to compose vectorial semantics in syntactic context. This is abstractly true in Smolensky's (1990) framework linking distributed, connectionist representations to the power of symbolic expressions by using tensor products. An example symbolic representation was to bind vectors to left- or right- child nodes in a binary (sub)tree structure; if vectors are instantiated with vector-semantic

meaning, then these bindings would precisely carry structured vectorial semantic representations of meaning. Indeed, Smolensky's work directly inspired SVS; however, simplifications (block-diagonal assumptions) are made in SVS to remove the need for more-complicated tensor products.

More practically, Kintsch (2001) modeled the subject-predicate relationship by composing LSA vectors for each word. As with information retrieval approaches, he used a weighted centroid for vector composition; nearest-neighbors of the target word were also included to come up with a 'gist' sense of meaning. The composed vectors performed well on several hand-selected similarity judgments. Mitchell and Lapata's work (2008), as described above, generalized the composition between vectors and formalized the evaluation, showing the benefits of multiplicative composition.

Mitchell and Lapata's subsequent work (Mitchell and Lapata, 2009) cast probabilities into the vectors, such that each vector element was a normalized likelihood $v_i = P(\text{context}_i|\text{target})/P(\text{context}_i)$, and a history (i.e., a context extending n-grams or incremental parser representations to include vectors) could be incrementally composed by a modified dot product. These vectors were then integrated with n-gram language models or interpolated with Roark's (2001) parser. The resulting language models are evaluated on perplexity, showing that the inclusion of these vectors better predicts the next word than an n-gram model or Roark's parser alone. This approach is much like SVS and incremental SVS: probabilities are also encapsulated in vectors, and incremental compositions are created. But SVS is a unified, generative model; therefore, incrementally composed semantics are not the only output — SVS necessarily outputs full-fledged parses as well. In contrast, Mitchell and Lapata's vector-based language models are interpolated with a parser, since the syntactic context considered at each word-vector does not correspond to a parse tree.

One technique related to these approaches may be confused with Structured Vectorial Semantics due to a naming overlap; Erk and Padó (2008) term their approach a 'structured vector space' (SVS) model. A 'structured vector space' is a model of lexical meaning that encodes the selectional preferences of a target word as well as a (possibly underspecified) vectorial representation of the word itself. Selectional preferences may be considered to answer: what might be arguments (or some other relation) of the target word, or what might the target word be an argument (or some other

relation) of? Different word classes will have different distributions observable by the context (Merlo and Stevenson, 2001). Because representations are being constructed for target words in the 'structured vector space,' relationships are also modeled as vectors; component-wise products are taken to compose vectors, following Mitchell and Lapata's observation (Mitchell and Lapata, 2008). The result is a context-specific construction of meaning, rather than a single composed vector. Structured Vectorial Semantics, on the other hand, composes lone vectorial representations for each node in hypothesized phrase structure, rather than just for individual words. Relations are also present in Structured Vectorial Semantics, but they are represented by matrices rather than vectors, and are therefore transformations rather than mere weightings.

9.4 Interpretation of Natural Language

As we have observed, combining vectorial semantics with full-fledged constituent parsing in a unified language model is a difficult problem, and SVS is unique in attempting to solve it. However, a multitude of other semantic representations, usually a logical form or formal language, have been used for natural language understanding. Many such approaches have been introduced in the field of semantic parsing, which includes the simpler task of semantic role labeling as well as more complicated interplays of syntax and semantics.

In an approach to deep-semantic parsing, Ge and Mooney (2005) map natural language sentences into a complete, formal, meaning representation language (MRL). These MRLs are defined by their domain; evaluations are done with CLang (from RoboCup competitions) and Geoquery (a query language for a geographical database). Implemented in the Scissor system, they use a single statistical model for syntax and semantics. After manually constructing and training on semantically-annotated parse trees, full constituent parsing is done with Collins' (1997) Model 2 head-driven parser.

This combination of full parsing and semantic representations in a unified language model, along with the semantic annotation of syntactic structure, are in line with the interactive nature of SVS. But SVS makes a conditional independence assumption that

approximately divides between the syntactic and semantic components at each expansion. Aside from this difference, a modified form of SVS[1] could in principle be used to produce the same semantic parses as SCISSOR in a specified domain. An additional mapping step gives the resulting semantics in the target MRL.

One weakness of the above approach is the high degree of supervision necessary for the manually-constructed semantically-annotated parse trees. Later work (Ge and Mooney, 2009) learned semantic correspondences automatically and modularized the syntactic portion to allow for other parsers, but other approaches have been considered as well. Instead of learning predicates from word-alignments, Liang, Jordan, and Klein (2009) focus on grounding meaning to a shared world model.

These approaches are designed to discover semantic information alongside syntax, and are thus more cognitively plausible as language models than parsers alone. However, similar to most of the approaches in standard constituent parsing, they are not designed as cognitive models and therefore lack such psycholinguistic qualities as interactivity, incrementality, and memory-bounded structural representations.

Work in machine reading and spoken language interfaces have attempted to model interpretation according to theories of human language processing. Martin and Riesbeck (1986) model interactive interpretation as a memory-accessing problem. They thus construct a large, well-indexed semantic representation in order to incrementally find broad-coverage meaning during parsing. Their model of syntax resides as linguistic structures stored completely within episodic memory (Direct Memory Access Parsing). Later work on this model (Livingston and Riesbeck, 2007; Forbus et al., 2007) leverages the existing knowledge to improve the acquisition and integration new information. Though incremental SVS is amenable to the dynamic integration of newly acquired information into an existing knowledge base, it models parsing as occurring in short-term memory rather than episodic memory.

Dynamic context may be considered to be a subset of the psycholinguistic goal of incremental processing. But incremental accounts of language processing are not just cognitively plausible; Aist et al. (2007) show that incremental interpretation actually improves both performance and usability of a spoken language interface.

[1] Syntactically, SVS may be used with head-driven parsing by modifying the composition equation in Chapter 3 to account for all left or right children. Semantically, elements of sets in Chapter 5 could include reified entities encoding for the domain of the particular MRL.

Incrementality and the dynamic nature of context go hand-in-hand in DeVault and Stone's model-theoretic dialogue interpreters (DeVault and Stone, 2003; DeVault and Stone, 2004). They use a factored model to model the pragmatics of underspecified logical forms, and treat interpretation as a constraint satisfaction problem.

He and Young (2005) encode dynamic local context as a 'Hidden Vector State' in a time-series model very similar to the Hierarchic Hidden Markov Model. Assuming right-branching structure (because the default in English is right-branching), they are able to construct hierarchical semantic representations.

Schuler, Wu, and Schwartz (2009) may be considered a more robust version of this idea, presenting a referential semantic language model that is a precursor to incremental SVS. They define an incremental spoken language interface that signals changes in a grounded world model, while simultaneously generating full syntactic parses. The syntactic portion is a modified version of Schuler et al. (2008; 2010), and therefore this model recognizes left-branching and center-embedded structure as well as right-branching structure, unlike He and Young's approach. Both syntax and surface-semantics are weighed together in a model of short-term memory, but episodic memory is maintained as supporting concept ontologies.

Incremental SVS subsumes the referential semantic language model. Unlike most vector-space systems, it is a framework general enough to be able to do this type of logical or model-theoretical interpretation, as shown in Chapter 5. In fact, future work on SVS may even be used to combine specific model-theoretic individuals with distributed, vector-space properties.

9.5 Summary

This chapter has overviewed work in distributed semantic spaces and constituent parsing to show how Structured Vectorial Semantics extends and combines some of the ideas from each respective area. Approaches that integrate syntax and semantics were also discussed in comparison to SVS (especially relationally-clustered SVS); SVS is unique in that it is a full-fledged parser along with semantic interpreter. Finally, work in machine reading and incremental interpretation were given, and the psycholinguistic assumptions of the models analyzed in comparison to incremental SVS.

Chapter 10

Conclusion

The main contribution of this thesis is *Structured Vectorial Semantics* (SVS), a generative framework that fully unifies vectorial semantics with syntactic context. Previous vector-space models of semantics have ignored syntax and used a bag-of-words assumption to capture useful topics from document-level relationships; SVS requires syntactic context and can capture local relationships. Semantic composition in vector-space models typically uses additive or multiplicative strategies; SVS uses a syntax-dependent linear operator chain. Attempts to unify vectorial semantics with parsing have mostly been explored in the context of dependency parsing; SVS produces full constituent parses. These characteristics of SVS show the uniqueness of the *structured* nature of SVS with respect to other vectorial semantic models.

The definition of SVS as a framework (Chapter 3) allowed for different instantiations of the model. Since vector-space models are typically built from dimensionality-reduced co-occurrence matrices, relational-clustering SVS was defined as this type of a vector space (Chapter 4). This model found most likely 'gist'-meaning semantic vectors while parsing with improved accuracy over a syntax-only baseline. Another instantiation of the framework was logical-interpretation SVS (Chapter 5). This instantiation of SVS connects the framework to a rich history in which formal logic can be used to handle meaning from the semantic level to the discourse level.

While Part I dealt mainly with the engineering implications of SVS, Part II focused on the cognitive and linguistic principles that inspired SVS and that drive further development. Distributed semantics in vectorial representations were likened to a 'family

resemblance' style of meaning. Meanings were refined based on dynamic context, and they contributed to that context. Much time was spent on making SVS *incremental* by casting it into an HHMM parser — which brought other psycholinguistic modeling assumptions in as well, such as probabilistically-ranked parallel hypotheses in interactive interpretation. But it was the bounded-memory characteristic of HHMMs, likened to short-term memory limits in humans, that combined with SVS to predict the distribution of some quantifiers in English.

The work of this thesis, and Structured Vectorial Semantics in particular, was defined such that it could be built upon and extended. In the realm of cognitively-plausible language modeling, distributed semantic vectors could be combined with vectors of precise definitions. Or, SVS would interface very nicely with a separate model of episodic memory; with SVS correctly augmented, problems such as semantic role labeling, coreference resolution, and anaphoric discourse relations would fall out naturally from this model. For use in resource-poor languages and other multi-lingual contexts, problems such as grammar induction or semantic estimation from dictionaries could be integrated with SVS. Finally, SVS could be applied to specific domains such as medical informatics, document classification, document summary, or spoken language interfaces.

My hope is that the contents of this thesis will inspire further exploration into interactive, cognitively-plausible models of language that are practically beneficial.

References

[Abney and Johnson1991] Abney, Steven P. and Mark Johnson. 1991. Memory requirements and local ambiguities of parsing strategies. *J. Psycholinguistic Research*, 20(3):233–250.

[Aist et al.2007] Aist, Gregory, James Allen, Ellen Campana, Carlos Gallo, Scott Stoness, Mary Swift, and Michael Tanenhaus. 2007. Incremental understanding in human-computer dialogue and experimental evidence for advantages over nonincremental methods. In *Proc. DECALOG*, pages 149–154.

[Alshawi1992] Alshawi, Hiyan, editor. 1992. *The core language engine*. MIT Press, Cambridge, MA.

[Altmann and Steedman1988] Altmann, G. and M. Steedman. 1988. Interaction with context during human sentence processing. *Cognition*, 30(3):191–238.

[Banerjee, Basu, and Merugu2007] Banerjee, Arindam, Sugato Basu, and Srujana Merugu. 2007. Multi-way clustering on relation graphs. In *Proceedings of the SIAM International Conference on Data Mining (SDM-2007)*, April.

[Bellman1957] Bellman, Richard. 1957. *Dynamic Programming*. Princeton University Press, Princeton, NJ.

[Bever1970] Bever, Thomas G.˙ 1970. The cognitive basis for linguistic structure. In J.R̃. Hayes, editor, *Cognition and the Development of Language*. Wiley, New York, pages 279–362.

[Bikel2004] Bikel, Daniel M. 2004. Intricacies of Collins' parsing model. *Computational Linguistics*, 30(4):479–511.

[Blei and McAuliffe2008] Blei, David M. and Jon D. McAuliffe. 2008. Supervised topic models. *Advances in Neural Information Processing Systems*, 20:121–128.

[Blei, Ng, and Jordan2003] Blei, David M., Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

[Bos1996] Bos, Johan. 1996. Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pages 133–143.

[Boston et al.2008] Boston, Marisa Ferrara, John T. Hale, Reinhold Kliegl, Umesh Patil, and Shravan Vasishth. 2008. Parsing costs as predictors of reading difficulty: An

evaluation using the Potsdam Sentence Corpus. *Journal of Eye Movement Research*, 2(1):1–12.

[Boyd-Graber and Blei2008] Boyd-Graber, Jordan and David M. Blei. 2008. Syntactic topic models. In *Neural Information Processing Systems*.

[Brants1999] Brants, Thorsten. 1999. Cascaded markov models. In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, page 125. Association for Computational Linguistics.

[Brants and Crocker2000] Brants, Thorsten and Matthew Crocker. 2000. Probabilistic parsing and psychological plausibility. In *Proceedings of COLING '00*, pages 111–118.

[Brown-Schmidt, Campana, and Tanenhaus2002] Brown-Schmidt, Sarah, Ellen Campana, and Michael K. Tanenhaus. 2002. Reference resolution in the wild: Online circumscription of referential domains in a natural interactive problem-solving task. In *Proceedings of the 24th Annual Meeting of the Cognitive Science Society*, pages 148–153, Fairfax, VA, August.

[Charniak1997] Charniak, Eugene. 1997. Statistical parsing with a context-free grammar and word statistics. In *Fourteenth National Conference on Artificial Intelligence*, Providence, Rhode Island.

[Charniak2000] Charniak, Eugene. 2000. A maximum-entropy inspired parser. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics (ANLP-NAACL'00)*, pages 132–139, Seattle, Washington.

[Charniak and Johnson2005] Charniak, Eugene and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, page 180. Association for Computational Linguistics.

[Church1940] Church, Alonzo. 1940. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68.

[Collins1996] Collins, Michael. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL '96)*.

[Collins1997] Collins, Michael. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL '97)*.

[Collins1999] Collins, Michael. 1999. *Head-driven statistical models for natural language parsing*. Ph.D. thesis, University of Pennsylvania.

[Collins2003] Collins, Michael. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637.

[Cowan2001] Cowan, Nelson. 2001. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24:87–185.

[Crain and Steedman1985] Crain, Stephen and Mark Steedman. 1985. On not being led up the garden path: The use of context by the psychological syntax processor. In D. R. Dowty, L. Karttunen, and A. M. Zwicky, editors, *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*, number 1 in Studies in Natural Language Processing. Cambridge University Press, Cambridge, pages 320–358.

[Dahan and Gaskell2007] Dahan, Delphine and M. Gareth Gaskell. 2007. The temporal dynamics of ambiguity resolution: Evidence from spoken-word recognition. *Journal of Memory and Language*, 57(4):483–501.

[Deerwester et al.1990] Deerwester, Scott, Susan Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

[Demberg and Keller2008] Demberg, Vera and Frank Keller. 2008. Data from eye-tracking corpora as evidence for theories of syntactic processing complexity. *Cognition*, 109(2):193–210.

[Dempster, Laird, and Rubin1977] Dempster, Arthur, Nan Laird, and Donald Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39 (Series B):1–38.

[DeVault and Stone2003] DeVault, David and Matthew Stone. 2003. Domain inference in incremental interpretation. In *Proc. ICoS*, pages 73–87.

[DeVault and Stone2004] DeVault, David and Matthew Stone. 2004. Interpreting vague

utterances in context. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING '04)*, pages 1247–1253.

[Eisner1996] Eisner, Jason. 1996. Three new probabilistic models for dependency grammar. In *Proceedings of the Sixteenth International Conference on Computational Linguistics (COLING '96).*

[Eisner and Satta1999] Eisner, Jason and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics*, pages 457–464. Association for Computational Linguistics.

[Ericsson and Kintsch1995] Ericsson, K. Anders and Walter Kintsch. 1995. Long-term working memory. *Psychological Review*, 102:211–245.

[Erk and Padó2008] Erk, Katrin and Sebastian Padó. 2008. A structured vector space model for word meaning in context. In *Proceedings of EMNLP 2008.*

[Fellbaum1998] Fellbaum, C. 1998. *WordNet: An Electronic Lexical Database.*

[Finkel, Grenager, and Manning2007] Finkel, Jerry R., Trond Grenager, and Christopher D. Manning. 2007. The infinite tree. In *Proceedings of the 45th Annual Meeting of the ACL*, volume 45, page 272.

[Forbus et al.2007] Forbus, Kenneth D., Christopher Riesbeck, Lawrence Birnbaum, Kevin Livingston, Abhishek Sharma, and Leo Ureel. 2007. Integrating natural language, knowledge representation and reasoning, and analogical processing to learn by reading. In *Proceedings of AAAI-07*, volume 22, page 1542.

[Ford, Bresnan, and Kaplan1982] Ford, M., J. Bresnan, and R. Kaplan. 1982. A competence-based theory of syntactic closure. *The mental representation of grammatical relations*, pages 727–796.

[Frank2009] Frank, Stefan L. 2009. Surprisal-based comparison between a symbolic and a connectionist model of sentence processing. In *Proc. Annual Meeting of the Cognitive Science Society*, pages 1139–1144.

[Ge and Mooney2005] Ge, Ruifang and Raymond J. Mooney. 2005. A statistical semantic parser that integrates syntax and semantics. In *Ninth Conference on Computational Natural Language Learning*, pages 9–16.

[Ge and Mooney2009] Ge, Ruifang and Raymond J. Mooney. 2009. Learning a compositional semantic parser using an existing syntactic parser. In *Proceedings of the*

47th Annual Meeting of the ACL, pages 611–619. Association for Computational Linguistics.

[Gibson1998] Gibson, Edward. 1998. Linguistic complexity: Locality of syntactic dependencies. *Cognition*, 68(1):1–76.

[Gildea2001] Gildea, Daniel. 2001. Corpus variation and parser performance. In *In Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP '00)*, Pittsburgh, PA.

[Gildea and Jurafsky2002] Gildea, Daniel and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3).

[Gorniak and Roy2003] Gorniak, Peter and Deb Roy. 2003. A visually grounded natural language interface for reference to spatial scenes. In *Proceedings of the International Conference for Multimodal Interfaces*.

[Gorniak and Roy2004] Gorniak, Peter and Deb Roy. 2004. Grounded semantic composition for visual scenes. *Journal of Artificial Intelligence Research*, 21:429–470.

[Grice1975] Grice, H.P. 1975. Logic and conversation. Academic Press, New York, pages 41–58.

[Griffiths and Steyvers2002] Griffiths, Thomas L. and Mark Steyvers. 2002. A probabilistic approach to semantic representation. In *Proceedings of the 24th annual conference of the cognitive science society*, pages 381–386. Citeseer.

[Griffiths et al.2005] Griffiths, Thomas L., Mark Steyvers, David M. Blei, and Joshua B. Tenenbaum. 2005. Integrating topics and syntax. *Advances in neural information processing systems*, 17:537–544.

[Groenendijk and Stokhof1991] Groenendijk, Jeroen and Martin Stokhof. 1991. Dynamic predicate logic. *Linguistics and Philosophy*, 14:39–100.

[Grosz and Sidner1986] Grosz, Barbara J. and Candace L. Sidner. 1986. Attention, intentions, and the structure of discourse. *Computational linguistics*, 12(3):175–204.

[Grosz, Weinstein, and Joshi1995] Grosz, B.J., S. Weinstein, and A.K. Joshi. 1995. Centering: a framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2):203–225.

[Gundel, Hedberg, and Zacharski1993] Gundel, Jeannette K., Nancy Hedberg, and Ron Zacharski. 1993. Cognitive status and the form of referring expressions in discourse. *Language*, 69(2):274–307.

[Hale2003] Hale, John. 2003. *Grammar, Uncertainty and Sentence Processing*. Ph.D. thesis, Cognitive Science, The Johns Hopkins University.

[Harris1954] Harris, Zellig. 1954. Distributional structure. *Word*, 10:146–162.

[He and Young2005] He, Yulan and Steve Young. 2005. Semantic processing using the hidden vector state model. *Computer speech & language*, 19(1):85–106.

[Heim1982] Heim, I. 1982. The semantics of definite and indefinite NPs. *University of Massachusetts at Amherst dissertation*.

[Hobbs1979] Hobbs, J.R. 1979. Coherence and coreference. *Cognitive Science*, 3(1):67–90.

[Hofmann1999] Hofmann, Thomas. 1999. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM.

[Hofmann2001] Hofmann, Thomas. 2001. Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning*, 42(1):177–196.

[Johnson1998] Johnson, Mark. 1998. Finite state approximation of constraint-based grammars using left-corner grammar transforms. In *Proceedings of COLING/ACL*, pages 619–623, Montreal, Canada.

[Johnson-Laird1987] Johnson-Laird, Philip N. 1987. The mental representation of the meaning of words. *Cognition*, 25(1-2):189–211.

[Johnson-Laird1994] Johnson-Laird, Philip N. 1994. Mental models and probabilistic thinking. *Cognition*, 50(1-3):189–209.

[Jurafsky1996] Jurafsky, Daniel. 1996. A probabilistic model of lexical and syntactic access and disambiguation. *Cognitive Science: A Multidisciplinary Journal*, 20(2):137–194.

[Kate2008] Kate, Rohit J. 2008. Transforming meaning representation grammars to improve semantic parsing. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 33–40. Association for Computational Linguistics.

[Kintsch1988] Kintsch, Walter. 1988. The role of knowledge in discourse comprehension: A construction-integration model. *Psychological review*, 95(2):163–182.

[Kintsch2001] Kintsch, Walter. 2001. Predication. *Cognitive Science*, 25(2):173–202.

[Klein and Manning2003] Klein, Dan and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan.

[Koo, Carreras, and Collins2008] Koo, Terry, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of the 46th Annual Meeting of the ACL*, volume 8. Citeseer.

[Koo and Collins2005] Koo, Terry and Michael Collins. 2005. Hidden-variable models for discriminative reranking. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, page 514. Association for Computational Linguistics.

[Kulkarni and Pedersen2005] Kulkarni, Anagha and Ted Pedersen. 2005. SenseClusters: unsupervised clustering and labeling of similar contexts. *Interactive Poster and Demonstration Session of the 43rd Annual Meeting of the ACL*, page 105.

[Landauer and Dumais1997] Landauer, T.K. and S.T. Dumais. 1997. A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge. *Psychological Review*, 104:211–240.

[Levy2008] Levy, Roger. 2008. Expectation-based syntactic comprehension. *Cognition*, 106(3):1126–1177.

[Liang, Jordan, and Klein2009] Liang, Percy, Michael I. Jordan, and Dan Klein. 2009. Learning semantic correspondences with less supervision. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL*, pages 91–99. Association for Computational Linguistics.

[Lin1998] Lin, Dekang. 1998. Automatic retrieval and clustering of similar words. In *Proceedings of the 36th Annual Meeting of the ACL*, volume 36, pages 768–774. Association for Computational Linguistics.

[Livingston and Riesbeck2007] Livingston, Kevin and Christopher K. Riesbeck. 2007. Using episodic memory in a memory based parser to assist machine reading. In *Working notes, AAAI Spring Symposium on Machine Reading*, volume 12.

[Lund and Burgess1996] Lund, K. and C. Burgess. 1996. Producing High-Dimensional Semantic Spaces From Lexical Co-Occurrence. *Behavior Research Methods Instruments and Computers*, 28:203–208.

[MacDonald, Pearlmutter, and Seidenberg1994] MacDonald, Maryellen C., Neal J. Pearlmutter, and Mark S. Seidenberg. 1994. The lexical nature of syntactic ambiguity resolution. *Psychological Review*, 101(4):676–703.

[Magerman1995] Magerman, David. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL'95)*, pages 276–283, Cambridge, MA.

[Marslen-Wilson1975] Marslen-Wilson, William D. 1975. Sentence perception as an interactive parallel process. *Science*, 189(4198):226–228.

[Martin and Riesbeck1986] Martin, Charles and Christopher Riesbeck. 1986. Uniform parsing and inferencing for learning. In *Proceedings of AAAI*, pages 257–261.

[Matsuzaki, Miyao, and Tsujii2005] Matsuzaki, Takuya, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 75–82. Association for Computational Linguistics.

[Merlo and Stevenson2001] Merlo, Paola and Suzanne Stevenson. 2001. Automatic verb classification based on statistical distributions of argument structure. *Computational Linguistics*, 27(3):373–408.

[Miller and Chomsky1963] Miller, George and Noam Chomsky. 1963. Finitary models of language users. In R. Luce, R. Bush, and E. Galanter, editors, *Handbook of Mathematical Psychology*, volume 2. John Wiley, pages 419–491.

[Miller1956] Miller, George A. 1956. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97.

[Mitchell and Lapata2008] Mitchell, Jeff and Mirella Lapata. 2008. Vector-based models of semantic composition. In *Proceedings of ACL-08: HLT*, pages 236–244, Columbus, OH.

[Mitchell and Lapata2009] Mitchell, Jeff and Mirella Lapata. 2009. Language Models Based on Semantic Composition. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 430–439.

[Montague1973] Montague, Richard. 1973. The proper treatment of quantification in

ordinary English. In J. Hintikka, J.M.E. Moravcsik, and P. Suppes, editors, *Approaches to Natural Langauge*. D. Riedel, Dordrecht, pages 221–242. Reprinted in R. H. Thomason ed., *Formal Philosophy*, Yale University Press, 1994.

[Mooney2007] Mooney, Raymond. 2007. Learning for semantic parsing. *Computational Linguistics and Intelligent Text Processing*, pages 311–324.

[Murphy and Paskin2001] Murphy, Kevin P. and Mark A. Paskin. 2001. Linear time inference in hierarchical HMMs. In *Proc. NIPS*, pages 833–840, Vancouver, BC, Canada.

[Nivre2007] Nivre, Joakim. 2007. Inductive dependency parsing. *Computational Linguistics*, 33(2).

[Nivre et al.2007] Nivre, Joakim, Johan Hall, Sandra K
"ubler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, volume 7, pages 915–932.

[Padó and Lapata2007] Padó, Sebastian and Mirella Lapata. 2007. Dependency-based construction of semantic space models. *Computational Linguistics*, 33(2):161–199.

[Pedersen and Bruce1997] Pedersen, Ted and Rebecca Bruce. 1997. Knowledge lean word-sense disambiguation. In *Proceedings of the National Conference on Artificial Intelligence*, pages 814–814.

[Petrov et al.2006] Petrov, Slav, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL'06)*.

[Petrov and Klein2007] Petrov, Slav and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of NAACL HLT 2007*, pages 404–411.

[Petrov and Klein2008] Petrov, Slav and Dan Klein. 2008. Sparse multi-scale grammars for discriminative latent variable parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 867–876. Association for Computational Linguistics.

[Pynte, New, and Kennedy2008] Pynte, J., B. New, and A. Kennedy. 2008. On-line contextual influences during reading normal text: A multiple-regression analysis. *Vision research*, 48(21):2172–2183.

[Rabiner1990] Rabiner, Lawrence R. 1990. A tutorial on hidden Markov models and selected applications in speech recognition. *Readings in speech recognition*, 53(3):267–296.

[Ramsey and Pfeffer2002] Ramsey, Normal and Avi Pfeffer. 2002. Stochastic lambda calculus and monads of probability distributions. In *Principles of Programming Languages (POPL'02)*, pages 154–165.

[Reisinger and Mooney2010] Reisinger, J. and R.J. Mooney. 2010. Multi-Prototype Vector-Space Models of Word Meaning.

[Roark2001] Roark, Brian. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

[Roark et al.2009] Roark, Brian, Asaf Bachrach, Carlos Cardenas, and Christophe Pallier. 2009. Deriving lexical and syntactic expectation-based measures for psycholinguistic modeling via incremental top-down parsing. *Proceedings of the 2009 Conference on Empirical Methods in Natural Langauge Processing*, pages 324–333.

[Roberts2004] Roberts, C. 2004. Context in dynamic interpretation. *Handbook of Contemporary Pragmatic Theory*, pages 197–220.

[Schuler2009] Schuler, William. 2009. Parsing with a bounded stack using a model-based right-corner transform. In *Proceedings of the North American Association for Computational Linguistics (NAACL '09)*, pages 344–352, Boulder, Colorado.

[Schuler et al.2008] Schuler, William, Samir AbdelRahman, Tim Miller, and Lane Schwartz. 2008. Toward a psycholinguistically-motivated model of language. In *Proceedings of COLING*, pages 785–792, Manchester, UK, August.

[Schuler et al.2010] Schuler, William, Samir AbdelRahman, Tim Miller, and Lane Schwartz. 2010. Broad-coverage incremental parsing using human-like memory constraints. *Computational Linguistics*, 36(1).

[Schuler, Wu, and Schwartz2009] Schuler, William, Stephen Wu, and Lane Schwartz. 2009. A framework for fast incremental interpretation during speech decoding. *Computational Linguistics*, 35(3):313–343.

[Schütze1998] Schütze, Hinrich. 1998. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–123.

[Schwartz et al.2009] Schwartz, Lane, Luan Nguyen, Andrew Exley, and William Schuler.

2009. In *Proceedings of the 2009 International Conference on Intelligent User Interfaces (IUI'09)*, pages 217–226, Sanibel Island, FL.

[Smolensky and Legendre2006] Smolensky, P. and G. Legendre. 2006. *The Harmonic Mind: From Neural Computation to Optimality-Theoretic Grammar*. MIT Press.

[Smolensky1990] Smolensky, Paul. 1990. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216.

[Steyvers and Griffiths2007] Steyvers, Mark and Thomas L. Griffiths. 2007. Probabilistic topic models. *Handbook of Latent Semantic Analysis*, pages 424–440.

[Surdeanu et al.2008] Surdeanu, Mihai, Richard Johansson, Adam Meyers, Lluìs Màrquez, and Joakim Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 159–177. Association for Computational Linguistics.

[Tanenhaus et al.1995] Tanenhaus, Michael K., Michael J. Spivey-Knowlton, Kathy M. Eberhard, and Julie E. Sedivy. 1995. Integration of visual and linguistic information in spoken language comprehension. *Science*, 268:1632–1634.

[Tarski1933] Tarski, Alfred. 1933. The concept of truth in the languages of the deductive sciences (polish). *Prace Towarzystwa Naukowego Warszawskiego, Wydzial III Nauk Matematyczno-Fizycznych*, 34. translated as 'The concept of truth in formalized languages', in: J. Corcoran (Ed.), Logic, Semantics, Metamathematics: papers from 1923 to 1938, Hackett Publishing Company, Indianapolis, IN, 1983, pp. 152–278.

[Taskar, Segal, and Koller2001] Taskar, Ben, Eran Segal, and Daphne Koller. 2001. Probabilistic classification and clustering in relational data. In *IJCAI'01: Proceedings of the 17th international joint conference on Artificial intelligence*, pages 870–876, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Webber et al.2003] Webber, BOnnie, Matthew Stone, Aravind Joshi, and Alistair Knott. 2003. Anaphora and discourse structure. *Computational Linguistics*, 29(4):545–587.

[Wilson and Sperber2004] Wilson, Deirdre and Dan Sperber. 2004. Relevance Theory. *Handbook of Pragmatics. Oxford: Blackwell*, pages 607–632.

[Wittgenstein1953] Wittgenstein, L. 1953. Philosophical Investigations, trans. GEM Anscombe. *New York*.

[Wu et al.2010] Wu, Stephen, Asaf Bachrach, Carlos Cardenas, and William Schuler. 2010. Complexity metrics in an incremental right-corner parser. In *Proceedings of the 49th Annual Conference of the Association for Computational Linguistics*.

[Wu, Schwartz, and Schuler2008a] Wu, Stephen, Lane Schwartz, and William Schuler. 2008a. Exploiting referential context in spoken language interfaces for data-poor domains. In *Proc. International Conference on Intelligent User Interfaces (IUI'08)*, Canary Islands, Spain, January.

[Wu, Schwartz, and Schuler2008b] Wu, Stephen, Lane Schwartz, and William Schuler. 2008b. Referential semantic language modeling for data-poor domains. In *Proc. ICASSP*.

[Zwarts and Winter2000] Zwarts, J. and Y. Winter. 2000. Vector space semantics: a model-theoretic analysis of locative prepositions. *Journal of Logic, Language and Information*, 9(2):169–211.

Appendix A

Proofs and Derivations

A.1 Derivation of incremental SVS HHMM Probabilities

This section expands Equations 7.31–7.35 for incremental SVS, showing how they are derived from the syntactic HHMM proabilities in Equations 7.13–7.17. First, all syntactic c variables are changed to ce; matrices $\mathbf{E}$ appear wherever a probablity distribution had two different semantic concepts among the condition and target variables.

- Cross-element expansion (CEE):

$$\mathsf{P}_{\theta_{\text{G-CEE},d}}(\langle c_{\eta\iota}, c'_{\eta\iota}, \mathbf{E}_{\eta\iota\times\eta\iota}\rangle \,|\, \langle -, c_\eta, -\rangle) \tag{A.1}$$

$$\stackrel{\text{def}}{=} [\![x_{\eta\iota} = c'_{\eta\iota} = c_{\eta\iota}]\!]$$

$$\cdot\, [\![\mathbf{E}_{\eta\iota\times\eta\iota} = e_\eta \mapsto e'_{\eta\iota} \mapsto \sum_{e_{\eta\iota}} \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta e_\eta \overset{*}{\to} c_{\eta\iota} e_{\eta\iota}\ \ldots) \cdot [\![e'_{\eta\iota} = e_{\eta\iota}]\!]]\!] \tag{A.2}$$

$$= [\![x_{\eta\iota} = c'_{\eta\iota} = c_{\eta\iota}]\!] \cdot [\![\mathbf{E}_{\eta\iota\times\eta\iota} = \mathbf{L}^*_{\eta\times\eta\iota}(c_\eta, c_{\eta\iota}) \cdot \mathbf{I}_{\eta\iota\times\eta\iota}]\!] \tag{A.3}$$

- Awaited transition (AWT):

$$\mathsf{P}_{\theta_{\text{G-AWT},d}}(\langle c_\eta, c_{\eta\iota 1}, \mathbf{E}_{\eta\times\eta\iota 1}\rangle \,|\, \langle c'_\eta, c_{\eta\iota}, \mathbf{E}_{\eta\times\eta\iota}\rangle\ c_{\eta\iota 0}\mathbf{e}_{\eta\iota 0})$$

$$\stackrel{\text{def}}{=} [\![c_\eta = c'_\eta]\!] \cdot [\![\mathbf{E}_{\eta\times\eta\iota 1} = e_\eta \mapsto e_{\eta\iota 1} \mapsto \textstyle\sum_{e_{\eta\iota}, e_{\eta\iota 0}} \mathbf{E}_{\eta\times\eta\iota}[e_\eta, e_{\eta\iota}] \cdot \mathbf{e}_{\eta\iota 0}[e_{\eta\iota 0}]$$

$$\cdot\, \frac{\mathsf{P}_{\theta_{\text{G-R},d}}(c_{\eta\iota} e_{\eta\iota} \to c_{\eta\iota 0} e_{\eta\iota 0}\ c_{\eta\iota 1} e_{\eta\iota 1})}{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_{\eta\iota} e_{\eta\iota} \overset{0}{\to} c_{\eta\iota 0} e_{\eta\iota 0}\ \ldots)}]\!] \tag{A.4}$$

$$= [\![c_\eta = c'_\eta]\!] \cdot [\![\mathbf{E}_{\eta\times\eta\iota 1} = e_\eta \mapsto e_{\eta\iota 1} \mapsto \textstyle\sum_{e_{\eta\iota}} \mathbf{E}_{\eta\times\eta\iota}[e_\eta, e_{\eta\iota}]$$
$$\cdot \textstyle\sum_{l_{\eta\iota 0}, l_{\eta\iota 1}} \mathsf{P}_{\theta_{\text{M-R},d}}(c_{\eta\iota} e_{\eta\iota} \to l_{\eta\iota 0} c_{\eta\iota 0} \;\; l_{\eta\iota 1} c_{\eta\iota 1})$$
$$\cdot \textstyle\sum_{e_{\eta\iota 0}} \mathsf{P}_{\theta_{\text{L}}}(e_{\eta\iota 0} \,|\, l_{\eta\iota 0} \; e_{\eta\iota}) \cdot \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_{\eta\iota} e_{\eta\iota} \overset{0}{\to} c_{\eta\iota 0} e_{\eta\iota 0} \; ...)^{-1}$$
$$\cdot \mathbf{e}_{\eta\iota 0}[e_{\eta\iota 0}] \cdot \mathsf{P}_{\theta_{\text{L}}}(e_{\eta\iota 1} \,|\, l_{\eta\iota 1} \; e_{\eta\iota})]\!] \tag{A.5}$$
$$= [\![c_\eta = c'_\eta]\!] \cdot [\![\mathbf{E}_{\eta\times\eta\iota 1} = \mathbf{E}_{\eta\times\eta\iota} \cdot \textstyle\sum_{l_{\eta\iota 0}, l_{\eta\iota 1}} \mathbf{M}_{\text{R},d}(c_{\eta\iota} \to l_{\eta\iota 0} c_{\eta\iota 0} \; l_{\eta\iota 1} c_{\eta\iota 1})$$
$$\cdot \mathrm{d}(\mathbf{L}_{\eta\iota\times 0}(l_0) \cdot \mathbf{L}^0_{\eta\iota\times\eta\iota 0}(c_{\eta\iota}, c_{\eta\iota 0})^{-1} \cdot \mathbf{e}_{\eta\iota 0}) \cdot \mathbf{L}_{\eta\iota\times 1}(l_1)]\!] \tag{A.6}$$

- Active transition (ACT):

$$\mathsf{P}_{\theta_{\text{G-ACT},d}}(\langle c_{\eta\iota}, c_{\eta\iota 1}, \mathbf{E}_{\eta\iota\times\eta\iota 1}\rangle \,|\, \langle -, c_\eta, -\rangle \; c_{\eta\iota 0} \mathbf{e}_{\eta\iota 0})$$
$$\overset{\text{def}}{=} [\![\mathbf{E}_{\eta\iota\times\eta\iota 1} = e_\eta \mapsto e_{\eta\iota 1} \mapsto \textstyle\sum_{e_{\eta\iota}, e_{\eta\iota 0}} \mathbf{e}_{\eta\iota 0}[e_{\eta\iota 0}]$$
$$\cdot \frac{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta e_\eta \overset{*}{\to} c_{\eta\iota} e_{\eta\iota} \; ...) \cdot \mathsf{P}_{\theta_{\text{G-L},d}}(c_{\eta\iota} e_{\eta\iota} \to c_{\eta\iota 0} e_{\eta\iota 0} \; c_{\eta\iota 1} e_{\eta\iota 1})}{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta e_\eta \overset{+}{\to} c_{\eta\iota 0} e_{\eta\iota 0} \; ...)}]\!] \tag{A.7}$$
$$= [\![\mathbf{E}_{\eta\iota\times\eta\iota 1} = e_\eta \mapsto e_{\eta\iota 1} \mapsto \textstyle\sum_{e_{\eta\iota}} \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta e_\eta \overset{*}{\to} c_{\eta\iota} e_{\eta\iota} \; ...)$$
$$\cdot \textstyle\sum_{l_{\eta\iota 0}, l_{\eta\iota 1}} \mathsf{P}_{\theta_{\text{M-L},d}}(c_{\eta\iota} e_{\eta\iota} \to l_{\eta\iota 0} c_{\eta\iota 0} \;\; l_{\eta\iota 1} c_{\eta\iota 1})$$
$$\cdot \textstyle\sum_{e_{\eta\iota 0}} \mathsf{P}_{\theta_{\text{L}}}(e_{\eta\iota 0} \,|\, l_{\eta\iota 0} \; e_{\eta\iota}) \cdot \mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta e_\eta \overset{+}{\to} c_{\eta\iota 0} e_{\eta\iota 0} \; ...)^{-1}$$
$$\cdot \mathbf{e}_{\eta\iota 0}[e_{\eta\iota 0}] \cdot \mathsf{P}_{\theta_{\text{L}}}(e_{\eta\iota 1} \,|\, l_{\eta\iota 1} \; e_{\eta\iota})]\!] \tag{A.8}$$
$$= [\![\mathbf{E}_{\eta\iota\times\eta\iota 1} = \mathbf{L}^*_{\eta\times\eta\iota}(c_\eta, c_{\eta\iota}) \cdot \textstyle\sum_{l_{\eta\iota 0}, l_{\eta\iota 1}} \mathbf{M}_{\text{L},d}(c_{\eta\iota} \to l_{\eta\iota 0} c_{\eta\iota 0} \; l_{\eta\iota 1} c_{\eta\iota 1})$$
$$\cdot \mathrm{d}(\mathbf{L}_{\eta\iota\times 0}(l_0) \cdot \mathbf{L}^+_{\eta\times\eta\iota 0}(c_\eta, c_{\eta\iota 0})^{-1} \cdot \mathbf{e}_{\eta\iota 0}) \cdot \mathbf{L}_{\eta\iota\times 1}(l_1)]\!] \tag{A.9}$$

- Cross-element reduction (CER):

$$\mathsf{P}_{\theta_{\text{G-CER},d}}(c_{\eta\iota}, \mathbf{e}_{\eta\iota}, 1 \,|\, \langle -, c_\eta, -\rangle \; \langle c'_{\eta\iota}, c_{\eta\iota\kappa}, \mathbf{E}_{\eta\iota\times\eta\iota\kappa}\rangle)$$
$$\overset{\text{def}}{=} [\![\mathbf{e}_{\eta\iota} = e_\eta \mapsto \sum_{e_{\eta\iota}, e_{\eta\iota\kappa}} \frac{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta e_\eta \overset{0}{\to} c_{\eta\iota} e_{\eta\iota})}{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta e_\eta \overset{*}{\to} c_{\eta\iota} e_{\eta\iota})} \cdot \mathbf{E}_{\eta\iota\times\eta\iota\kappa}[e_{\eta\iota}, e_{\eta\iota\kappa}]]\!] \tag{A.10}$$
$$= [\![\mathbf{e}_{\eta\iota} = \mathbf{L}^0_{\eta\times\eta\iota}(c_\eta, c_{\eta\iota}) \cdot \mathbf{L}^*_{\eta\times\eta\iota}(c_\eta, c_{\eta\iota})^{-1} \cdot \mathbf{E}'_{\eta\iota\times\eta\iota\kappa} \cdot \mathbf{1}]\!] \tag{A.11}$$

- In-element reduction (IER):

$$\mathsf{P}_{\theta_{\text{G-CER},d}}(c_{\eta\iota}, \mathbf{e}_{\eta\iota}, 0 \,|\, \langle -, c_\eta, -\rangle \; \langle c'_{\eta\iota}, c_{\eta\iota\kappa}, \mathbf{E}_{\eta\iota\times\eta\iota\kappa}\rangle)$$
$$\overset{\text{def}}{=} [\![\mathbf{e}_{\eta\iota} = e_\eta \mapsto \sum_{e_{\eta\iota}, e'_{\eta\iota}, e_{\eta\iota\kappa}} \frac{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta e_\eta \overset{+}{\to} c_{\eta\iota} e_{\eta\iota})}{\mathsf{E}_{\theta_{\text{G-RL}^*,d}}(c_\eta e_\eta \overset{*}{\to} c_{\eta\iota} e_{\eta\iota})} \cdot \mathbf{E}_{\eta\iota\times\eta\iota\kappa}[e_{\eta\iota}, e_{\eta\iota\kappa}]]\!] \tag{A.12}$$

$$= [\![\mathbf{e}_{\eta\iota} = \mathbf{L}^{+}_{\eta\times\eta\iota}(c_\eta, c_{\eta\iota}) \cdot \mathbf{L}^{*}_{\eta\times\eta\iota}(c_\eta, c_{\eta\iota})^{-1} \cdot \mathbf{E}'_{\eta\iota\times\eta\iota\kappa} \cdot \mathbf{1}]\!] \tag{A.13}$$