

Category-Theoretic Quantitative Compositional Distributional Models of Natural Language Semantics



Edward Grefenstette

Balliol College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2013

To my parents, Irene and Greg.

Acknowledgements

I would like to begin by thanking my supervisors, Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Pulman, for their guidance and help over the last three years. In particular, I thank Mehrnoosh for her invaluable support in the day-to-day aspects of my work, and for overseeing the production and publication of the papers we co-authored during the course of my DPhil.

On the academic front, I would also like to thank, in no particular order, Stephen Clark, Phil Blunsom, Marco Baroni, Georgiana Dinu, Yao-Zhong Zhang, Tom Melham, Samson Abramsky, Yorick Wilks, and the many other world-class researchers whom I have had the pleasure of interacting with and being challenged by as I learned the ropes of the research world. Thanks also go to the Engineering and Physical Sciences Research Council for funding my doctoral work through a Doctoral Training Account and an Enhanced Stipend.

From home, I thank my loving parents Irene and Greg for their emotional and material support over the last twenty-seven years, and for giving me the opportunity to reach this level of research (as well as proof-reading my dissertation). I thank Grandma and Grandpa Grefenstette for supporting my studies and personal development both at Oxford and throughout life. I also thank my maternal grandparents, Bernard and Olga, my brother Nicholas, my sister Natalie, and the rest of my family for all the love I've received over the years. This work would not have been possible without it.

Finally, thanks to my friends in and out of Oxford, and in particular to Chris, John, Renie, Kelsey, Tom and Emily, Sean, Jenn, Karl, and Jan, for putting up with me throughout the doctoral experience. Listing all those who have made the last few years of my life an adventure is beyond the scope of this thesis, but to all those I've omitted: thank you. It's been a blast.

Abstract

This thesis is about the problem of compositionality in distributional semantics. Distributional semantics presupposes that the meanings of words are a function of their occurrences in textual contexts. It models words as distributions over these contexts and represents them as vectors in high dimensional spaces. The problem of compositionality for such models concerns itself with how to produce distributional representations for larger units of text (such as a verb and its arguments) by composing the distributional representations of smaller units of text (such as individual words).

This thesis focuses on a particular approach to this compositionality problem, namely using the categorical framework developed by Coecke, Sadrzadeh, and Clark, which combines syntactic analysis formalisms with distributional semantic representations of meaning to produce syntactically motivated composition operations. This thesis shows how this approach can be theoretically extended and practically implemented to produce concrete compositional distributional models of natural language semantics. It furthermore demonstrates that such models can perform on par with, or better than, other competing approaches in the field of natural language processing.

There are three principal contributions to computational linguistics in this thesis. The first is to extend the DisCoCat framework on the syntactic front and semantic front, incorporating a number of syntactic analysis formalisms and providing learning procedures allowing for the generation of concrete compositional distributional models. The second contribution is to evaluate the models developed from the procedures presented here, showing that they outperform other compositional distributional models present in the literature. The third contribution is to show how using category theory to solve linguistic problems forms a sound basis for research, illustrated by examples of work on this topic, that also suggest directions for future research.

Contents

1	Introduction	1
I	Background	7
2	Literature Review	9
2.1	Formal Semantics	10
2.2	Distributional Semantics	11
2.3	Compositionality and Vector Space Models	15
2.3.1	Additive Models	15
2.3.2	Multiplicative Models	16
2.3.3	Mixture Models	17
2.3.4	Tensor-Based Models	18
2.3.4.1	Dimensionality Problems	19
2.3.4.2	Syntactic Expressivity	21
2.3.5	SVS Models	22
2.4	Matrix-Based Compositionality	24
2.4.1	Generic Additive Model	24
2.4.2	Adjective Matrices	25
2.4.3	Recursive Matrix-Vector Model	26
3	Foundations of DisCoCat	29
3.1	Pregroup Grammars	30
3.1.1	Pregroups	30
3.1.2	Pregroups and Syntactic Analysis	32
3.1.3	A graphical calculus for pregroups	32
3.2	Categories	33
3.2.1	The Basics of Category Theory	34
3.2.2	Compact Closed Categories	35
3.2.3	A Graphical Calculus for Compact Closed Categories	37
3.3	A Categorical Passage from Grammar to Semantics	39
3.3.1	FVect	41
3.3.2	Syntax Guides Semantics	42
3.3.3	Example	43

II	Theory	45
4	Syntactic Extensions	47
4.1	Functorial Passages	48
4.1.1	Functors	49
4.1.2	From Product Categories to Functors	51
4.2	Supporting Context Free Grammars	53
4.2.1	Context Free Grammar	54
4.2.1.1	General Definition	54
4.2.1.2	Restrictions	56
4.2.2	CFGs as Categories	59
4.2.3	Defining a Functor	61
4.2.3.1	CFG to Pregroup Translation	61
4.2.3.2	From Translation Dictionaries to Functors	66
4.3	Supporting Lambek Grammar	68
4.3.1	Lambek Grammar	68
4.3.2	Lambek Grammars as Monoidal Bi-Closed Categories	70
4.3.3	Defining a Functor	74
5	Learning Procedures for a DisCoCat	79
5.1	Defining Sentence Space	80
5.1.1	Intuition	80
5.1.2	A Concrete Proposal	81
5.2	Noun-Oriented Types	82
5.2.1	Dealing with Nouns	82
5.2.2	Dealing with Relational Words	82
5.3	Learning Procedures	82
5.3.1	Groundwork	82
5.3.2	A Learning Algorithm	83
5.3.3	Problems with Reduced Representations	85
5.3.4	A Generalised Algorithm for Reduced Representations	88
5.3.5	Example	89
5.3.6	An Efficient Alternative	90
III	Practice	95
6	Evaluating a DisCoCat	97
6.1	First Experiment	98
6.1.1	Dataset Description	98
6.1.2	Evaluation Methodology	100
6.1.3	Models Compared	101
6.1.4	Results	102
6.2	Second Experiment	103
6.2.1	Dataset Description	103

6.2.2	Evaluation Methodology	104
6.2.3	Models Compared	104
6.2.4	Results	105
6.3	Third Experiment	106
6.3.1	Dataset Description	106
6.3.2	Evaluation Methodology	106
6.3.3	Models Compared	107
6.3.4	Results	108
6.4	Discussion	109
7	Further Work	111
7.1	Distributional Logic	112
7.1.1	Distributional Formal Semantics	112
7.1.2	Tensors as Functions	113
7.1.3	Formal Semantics with Tensors	114
7.1.4	Simulating Simple Predicate Calculi	116
7.1.5	Logical Operations, and Integrating Non-Linearity	121
7.2	Learning Tensors by Multi-Step Linear Regression	127
7.2.1	Multi-Step Linear Regression	128
7.2.2	Experiments	130
7.2.3	Discussion	135
7.3	Further Syntactic Extensions: Combinatory Categorical Grammar	136
7.3.1	Combinatory Categorical Grammar	136
7.3.2	Categorical Semantics for CCGs	138
7.3.3	Defining a Functor	144
7.4	Next Steps	147
8	Conclusions	151
	Bibliography	153

List of Figures

2.1	A simple model of formal semantics.	10
2.2	A simple model of distributional semantics.	13
3.1	Diagrammatic examples of pregroup reductions.	33
3.2	Diagrammatic pregroup parse of “John loves Mary”.	33
3.3	Examples of yank-slide equalities in the graphical calculus for compact closed categories.	39
3.4	Sample diagrammatic representation of distributional composition.	42
4.1	Sample parse tree for the CFG in Table 4.1.	56
4.2	Procedure for translating a CFG with $ R_{NT} $ production rules into $k \leq 2^{ R_{NT} }$ pregroup grammars.	64
4.3	Basic diagrammatic language constructs.	71
4.4	Rewrite rules for ev^l and ev^r	72
4.5	Diagrams for currying rules.	73
4.6	Diagrams for composition rules.	73
4.7	Diagrams for type raising rules.	74
4.8	Diagrams for sample Lambek Grammar parses.	74
4.9	Diagrams for compact closed currying.	76
5.1	Diagrammatic form of reduced representations.	86
5.2	Composition under reduced representations.	87
5.3	Procedure for learning weights for matrices of words ‘P’ with relational types π of m arguments.	88
5.4	Composition under the Kronecker model.	92
5.5	Composition under the generalised Kronecker model.	93
7.1	A simple formal semantic model.	113
7.2	Estimating a tensor for <i>eat</i> in two steps.	130
7.3	Simple diagrammatic form of <i>cross</i> morphisms.	140
7.4	Simple diagrammatic form of <i>lsub</i> morphisms.	140
7.5	Simple diagrammatic form of <i>rsub</i> morphisms.	141
7.6	Diagrams for <i>lswap</i> and <i>rswap</i> morphisms.	141
7.7	Expanded diagrammatic form of <i>cross</i> morphisms.	142
7.8	Diagrammatic representation of μ morphisms.	142
7.9	Expanded diagrammatic form of <i>lsub</i> morphisms.	143

7.10 Expanded diagrammatic form of *rsub* morphisms. 144

List of Tables

3.1	Basic elements of the graphical calculus for compact closed categories. . .	38
3.2	Structural morphisms in the graphical calculus for compact closed categories.	39
3.3	Rewrite rules in the graphical calculus for compact closed categories. . . .	40
4.1	A simple CFG.	55
4.2	Axioms of Lambek Grammar.	69
5.1	Sample weights for selected noun vectors.	89
5.2	Sample semantic matrix for ‘show’.	90
6.1	Example entries from the intransitive dataset without annotator score, first experiment.	99
6.2	Model correlation coefficients with human judgements, first experiment. $p < 0.05$ for each ρ	103
6.3	Example entries from the transitive dataset without annotator score, second experiment.	104
6.4	Model correlation coefficients with human judgements, second experiment. $p < 0.05$ for each ρ	105
6.5	Example entries from the adjective-transitive dataset without annotator score, third experiment.	107
6.6	Model correlation coefficients with human judgements, third experiment. $p < 0.05$ for each ρ	109
7.1	Spearman correlation of composition methods with human similarity in- tuitions on two sentence similarity data sets (all correlations significantly above chance).	134

Chapter 1

Introduction

For a large class of cases—though not for all—in which we employ the word “meaning” it can be explained thus: the meaning of a word is its use in the language.

—Ludwig Wittgenstein

Language is fundamentally messy. Since the early days of philosophical thought, thinkers have sought to tease structure out of it, from the Aristotelean syllogisms to the development of mathematical logic in the work of Peano and Frege, *inter alia*, at the dawn of the twentieth century. Others, such as the later Wittgensteinian school of thought, held that language was perfectly ‘in order’ as it was, and that it is by measuring the correctness of our speech acts against the linguistic community we live in that we learn how to properly use language, rather than by appealing to some objective standard dictating the correct structure of meaning.

As the world evolved into the present age of information, language became not only the purview of linguists, philosophers and logicians, but also of engineers and scientists. The rapid increase of machine readable and publicly available text through the development of the world wide web prompted a need for new technological tools to systematically classify, search, translate, summarise, and analyse text. As this need grew, new mathematical and computational methods were developed to extract structure from unstructured data.

While some thinkers of the modern age sought to adapt the tools and methods of logicians, leading to the development of mathematical accounts of natural language meaning such as formal semantics, others wished to accept the messiness of language. They instead turned to the large amount of data available in order to model meaning through statistical and distributional means. Thus while formal semanticists treated natural language as a programming language, with grammar as its syntax and logic as its semantics, distributional

and statistical semanticists interpreted the meaning of our words as being a function of their contexts, as observed in text from various sources.

Both in the case of philosophers of language and computational linguists, there is an apparent mutual inconsistency between the formal and distributional/empirical approaches. They seem orthogonal, in that the former portrays language meaning as well-organised, tidy, and well structured; while the latter aims to learn the underlying meaning of words without appeal to any underlying structure beyond the superficial relations words hold to one another through grammatical relations, or through simply occurring within the same sentence or document. However, it could be argued that such a distinction between these two views of semantics forms a false dichotomy, and that there is some middle ground on which we could both think of language as being a matter of structured relations and functions on the one hand, and empirically learnable meaning on the other.

This thesis deals with this middle ground. In the past decade, computational linguists have sought to combine the structured approach of formal semantics and the empirical methods of distributional semantics to produce a new class of models of natural language semantics, dubbed *compositional distributional semantic models*, capable of exploiting the structured aspects of language while learning meaning empirically.

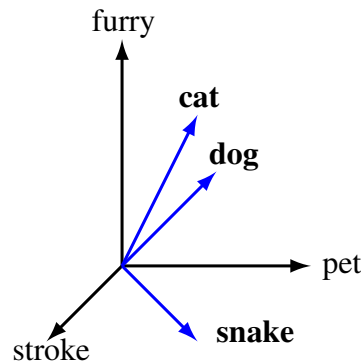
I begin, in Chapter 2, by giving an overview of these two seemingly different ways of modelling natural language semantics. I discuss formal semantics, which creates an association between grammatical elements and parts of logical expressions, in which syntactic analysis yields the production of logical statements based on how grammatical elements combine to form sentences.

Syntactic Analysis	Semantic Interpretation
$S \Rightarrow NP VP$	$\llbracket VP \rrbracket(\llbracket NP \rrbracket)$
$NP \Rightarrow \text{cats, milk, etc.}$	$\llbracket \text{cats} \rrbracket, \llbracket \text{milk} \rrbracket, \dots$
$VP \Rightarrow Vt NP$	$\llbracket Vt \rrbracket(\llbracket NP \rrbracket)$
$Vt \Rightarrow \text{like, hug, etc.}$	$\lambda yx. \llbracket \text{like} \rrbracket(x, y), \dots$

\Rightarrow

A simple formal semantic model.

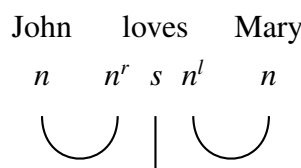
I then introduce distributional semantic models, which model the meaning of words as vectors in high dimensional spaces, constructed from the other words they co-occur with.



Graphical representation of a simple distributional semantic model.

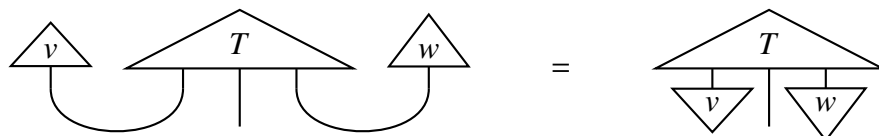
Furthermore, I discuss past attempts to reconcile both approaches with mathematical operations that compose word vectors to form semantic vectors for larger units of text such as phrases or sentences, surveying the different approaches to this problem in the literature.

In Chapter 3, I give an overview of an existing framework—DisCoCat—developed by [19], which provides a general account of how grammatical formalisms and distributional semantics can be combined to generate compositional distributional models of semantics. The DisCoCat framework borrows a powerful mathematical tool frequently used in quantum information theory, namely category theory, which allows information to be communicated between different mathematical formalisms, provided that they share some underlying structure. I survey the background knowledge required to understand this framework by introducing pregroup grammars, a syntactic formalism with a convenient algebraic structure, and which is easily represented as a category.



Pregroup parse of “John loves Mary”.

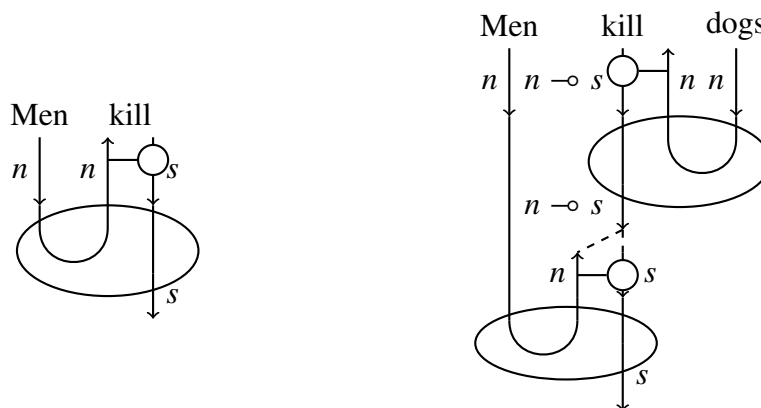
I also introduce the basics of category theory, and the diagrammatic calculus which allows us to reason about compositional operations within a category.



Graphical representation of subject-verb-object composition in DisCoCat.

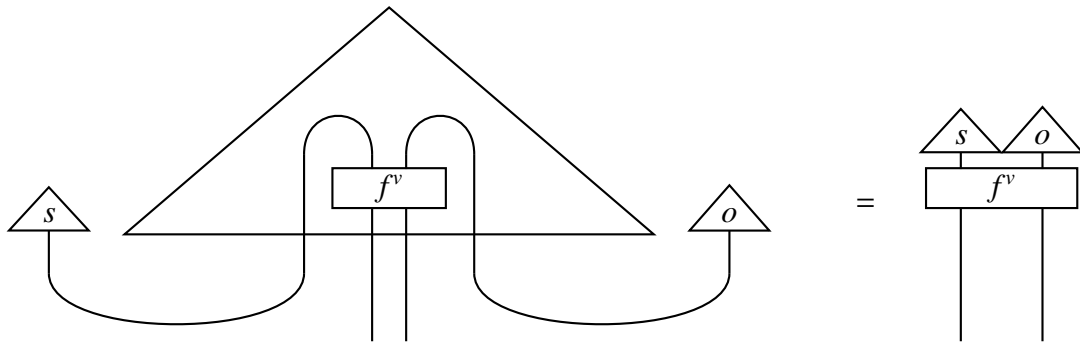
Finally, I discuss how distributional semantics fits into the categorical discourse within DisCoCat, and how giving categorical representations to both pregroup grammars and vector spaces allows us to produce syntactically motivated composition operations.

In Chapter 4, I develop new syntactic extensions to the DisCoCat framework. I begin by presenting the notion of a functorial passage, and discuss how functors can be used to extend the framework to incorporate syntactic formalisms other than pregroup grammars. I then illustrate this by showing how Context Free Grammars can be incorporated into the extended framework by translating them into pregroup grammars. I also show how Lambek Grammars can be integrated into the framework by interpreting them as a specific kind of category called a bi-closed monoidal category. I show how an existing graphical calculus for such categories can be applied to develop a functorial passage from these categories to the categories representing our semantic models.



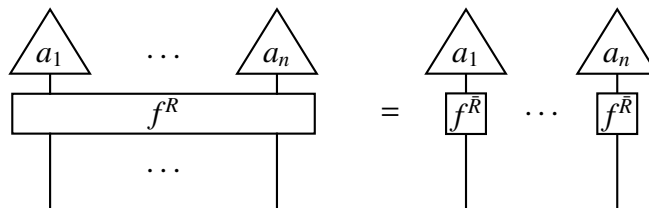
Example of the diagrammatic calculus for categorical representation of Lambek Grammar.

In Chapter 5, I present a new learning procedure for generating concrete compositional distributional semantic models from the DisCoCat framework. This procedure learns the semantic representations for relations by summing over the information that the relations' arguments hold in a corpus. I present a reduced representation for such relations which allows us to efficiently compute the semantic representations for relations which would otherwise require large amounts of data to model. I define composition operations for these reduced representations. Furthermore, I demonstrate that, under certain assumptions, the reduced representations are equivalent to the full representations of the relation they model and are embedded within these full representations.



Diagrammatic example of embedded reduced representations.

I furthermore provide a simplified alternative learning procedure for these reduced representations, which ignores some aspects of the relations it provides a model for, but is significantly quicker to learn and more efficient to store, while outperforming the model it supplants in certain experiments.



Diagrammatic form of generalised Kronecker composition model.

In Chapter 6, I evaluate the new models described in the previous chapter in a series of sentence similarity detection experiments, comparing these models with other compositional distributional models of semantics. These experiments test how well compositional models can produce sentence representations for sentences containing ambiguous words, and disambiguate the constituent words through the compositional process.

Sentence 1	Sentence 2
butler bow	butler submit
head bow	head stoop
company bow	company submit
government bow	government stoop

Sample sentence pairs from a phrase similarity detection experiment.

These experiments show that the models generated by the DisCoCat framework and the learning procedures developed in this thesis outperform other approaches to distributional compositionality.

Finally, in Chapter 7, I discuss additional work that I have done on this topic with various collaborators, and outline future work to continue to develop this growing field. Three further developments are described. The first addresses the issue of incorporating logical operations into compositional distributional models. I discuss how a simple predicate calculus might be simulated using tensors, and how this simulation fits into the DisCoCat framework, before considering some of the difficulties that arise when trying to model genuine logical calculi using distributional methods. Second, I discuss how machine learning methods such as linear regression may be adapted to the DisCoCat framework, as an alternative to the learning procedures I initially presented. Third, I provide the foundations for integrating a complex grammatical formalism, Combinatory Categorical Grammars, into the DisCoCat framework, and discuss issues faced when trying to accommodate the many variants this formalism possesses. I conclude by suggesting general directions future research might take based on these three points.

The aim of this thesis is to not only provide an exploration of the ways in which the DisCoCat framework can be extended, but to also give a proof-of-concept, showing that this abstract framework can be instantiated, that concrete models can be developed based on it, and that these models offer an interesting new direction in the search for sophisticated models of natural language semantics.

Part I

Background

Chapter 2

Literature Review

Chapter Abstract

This chapter presents the background literature concerning compositionality in distributional semantics. The reader will find an overview of another well-known compositional formalism, formal semantics, as well as a brief reminder of the basic concepts behind distributional semantic models. Then follows a survey of various approaches to the mathematics of vector composition from the last few decades, and the discussing and contrasting of various matrix-vector based approaches to distributional compositionality.

Compositional formal semantic models represent words as parts of logical expressions, composed according to grammatical structure. These models embody classical ideas in logic and philosophy of language, mainly Frege’s principle that the meaning of a sentence is a function of the meaning of its parts [31]. Well studied and robust, logical formalisms offer a scalable theory of meaning which can be used to reason about language using logical tools of proof and inference. In contrast, distributional models are a more recent approach to semantic modelling, representing the meaning of words not as logical formula but as vectors whose values have been empirically learned from corpora. Distributional models have found their way into real world applications such as thesaurus extraction [41, 21] or automated essay marking [55], and have strong connections to semantically motivated information retrieval [59]. This new dichotomy in defining properties of meaning: ‘logical form’ versus ‘contextual use’, has left the question of the foundational structure of meaning, initially of sole concern to linguists and to philosophers of language, even more of a challenge.

In this chapter, I present an overview of the background to the work developed in this thesis by briefly describing formal and distributional approaches to natural language seman-

tics, and providing a non-exhaustive list of some approaches to compositional distributional semantics. For a more complete review of the topic, I encourage the reader to consult one of the many excellent surveys of the field [80, 21, 12].

2.1 Formal Semantics

The approach commonly known as *Formal Semantics*, principally due to the work of Richard Montague in the area of inductive logic, provides methods for translating sentences of natural language into logical formulae, which can then be fed to computer-based reasoning tools [2].

To compute the meaning of a sentence consisting of n words, the individual meanings of each words must *interact*. In formal semantics, this interaction is represented as a function derived from the grammatical structure of the sentence. Formal models consist of a pairing of syntactic analysis rules (in the form of a grammar) with semantic interpretation rules, as exemplified by the simple model presented on the left hand side of Figure 2.1.

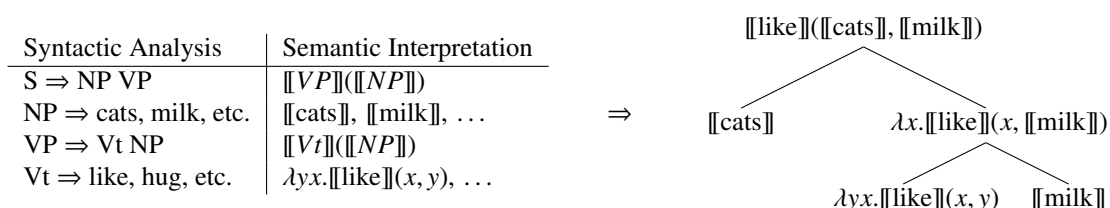


Figure 2.1: A simple model of formal semantics.

The semantic representations of words here are expressions of a higher-order logic formed by lambda expressions over parts of first-order logical formulae, which can be combined with one another to form well-formed expressions of first-order logic. The function $\llbracket - \rrbracket : \mathcal{L} \rightarrow \mathcal{M}$ maps elements of the lexicon \mathcal{L} to their interpretation in the logical model \mathcal{M} used. Proper nouns are typically just logical atoms, while nouns, adjectives, verbs, and other relational words are interpreted as predicates and relations. The parse of a sentence such as “cats like milk”, represented here as a binarised parse tree, is used to produce its semantic interpretation by substituting semantic representations for their grammatical constituents and applying β -reduction where needed. Such a derivation is shown on the right hand side of Figure 2.1.

What makes this class of models attractive is that it reduces language meaning to logical expressions, a subject well-known to philosophers of language, logicians, and linguists. The properties of first-order and higher-order logics are well studied, and it is possible

to evaluate the meaning of a sentence if given a logical model and domain (the model-theoretic approach to logic), as well as to verify whether or not one sentence entails another according to the rules of logical consequence and deduction, based on syntactic rules (the syntactic approach to logic).

However, such logical analysis says nothing about the closeness in meaning or topic of expressions beyond their truth-conditions and which models satisfy these truth conditions. Therefore, formal semantic approaches to modelling language meaning do not perform well on language tasks where the notion of similarity is not strictly based on truth conditions, such as document retrieval, topic classification, etc. Furthermore, an underlying domain of objects and a valuation function must be provided, as with any logic, leaving open the question of how we might *learn* the meaning of language using such a model, rather than just use it.

2.2 Distributional Semantics

The distributional semantics approach to lexical semantics represents the meaning of words as distributions in a high-dimensional vector space. This approach is based on the *distributional hypothesis* of Harris [44], who postulated that the meaning of a word was dictated by the context of its use. The more famous dictum stating this hypothesis is Firth’s statement [28] that “You shall know a word by the company it keeps”. This view of semantics has furthermore been associated [33, 80] with earlier work in philosophy of language by Wittgenstein, presented in [88], who stated that language meaning was equivalent to its real world use.

Practically speaking, in this approach, the meaning of a word can be learned from a corpus by looking at what other words occur with it within a certain *context*. The resulting distribution can be represented as a vector in a semantic vector space. This vectorial representation is convenient because vectors are a familiar structure with a rich set of ways of computing vector distance, allowing us to experiment with different word similarity metrics. The geometric nature of this representation entails that we can not only compare individual words’ meaning with various levels of granularity (e.g. we might, for example, be able to show that cats are closer to kittens than to dogs, but that all three are mutually closer than cats and steam engines), but also apply methods frequently called upon in information retrieval tasks such as those described by [59], to group concepts by topic, sentiment, or other semantic classes.

The distribution underlying word meaning here is a vector in a vector space, the basis vectors of which are dictated by the context. In simple models, the basis vectors will be

annotated with words from the lexicon. Traditionally, the vector spaces used in such models are Hilbert spaces, i.e. vector spaces with orthogonal bases, such that the inner product of any one basis vector with another (other than itself) is zero. The semantic vector for any word can be represented as the weighted superposition (i.e. the vector sum) of the basis vectors:

$$\overrightarrow{\text{some word}} = \sum_i c_i \vec{n}_i$$

where some set of orthogonal unit vectors $\{\vec{n}_i\}_i$ is the basis of the vector space which the meaning of the word lives in, and $c_i \in \mathbb{R}$ is the weight associated with basis vector \vec{n}_i .

A common source of confusion when encountering these models for the first time is to think of these annotated basis vectors as representations of the words which annotate them. In fact, they are simply representations of those words as context tokens. For example, the word ‘furry’ may be used to annotate one of the basis elements of a vector space in its role as a context. However, when we wish to reason about the meaning of ‘furry’, we will use the semantic vector associated with it in the vector space, which is distinct from the basis vector annotated with ‘furry’. This distinction may help explain why the orthogonality of the basis vectors is acceptable despite the fact that some of the words which annotate them may have similar meaning: we assume their context-theoretic properties to be independent, even if the semantic vectors we will eventually construct for these words may be closer in the space.

The construction of the vector for a word is done by counting, for each lexicon word n_i associated with basis vector \vec{n}_i , how many times n_i occurs in the context of each occurrence of the word for which we are constructing the vector. This count is then typically adjusted according to a weighting scheme (e.g. term frequency inverse document frequency). The “context” of a word can be something as simple as the other words occurring in the same sentence as the word or within k words of it, or something more complex, such as using dependency relations or other syntactic features.

To give an example, suppose that we wish to construct the semantic vectors for the meaning of ‘dog’, ‘cat’ and ‘snake’. Let us fix the set of possible context words to be {‘furry’, ‘pet’, ‘stroke’}. Let us consider ‘context’ to mean “words occurring in the same sentence as the target word”. We begin by looking at the instances of ‘dog’ in some training corpus of real text usage and see that it occurs twice with ‘furry’, twice with ‘pet’ and once with ‘stroke’. We therefore learn from the training corpus the vector:

$$\overrightarrow{\text{dog}} = [2 \ 2 \ 1]^T$$

We then observe that ‘cat’ occurs thrice in the same sentence as ‘furry’, once as ‘pet’ and does not occur in the same sentence as stroke. Likewise, we note that ‘snake’ does not occur in the context of ‘furry’, but twice in the context of ‘pet’ and ‘stroke’. We therefore build the vectors:

$$\vec{\text{cat}} = [3 \ 1 \ 0]^\top \quad \vec{\text{snake}} = [0 \ 2 \ 2]^\top$$

Figure 2.2 shows a graphical representation of the semantic space in which we built these vectors. We can visually observe that ‘cat’ and ‘dog’ seem closer in this space than ‘cat’ and ‘snake’ or ‘dog’ and ‘snake’. We can also see that ‘dog’ appears closer to ‘snake’ in this space than the vector for ‘cat’ is close to ‘snake’. The geometric notion of a semantic space therefore yields this non-binary aspect of similarity: concepts are not classified as similar or dissimilar, but are instead part of a continuum of similarity defined by the similarity metric used to compare vectors.

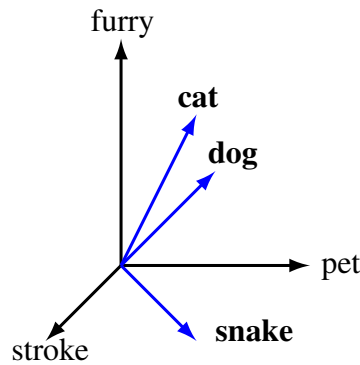


Figure 2.2: A simple model of distributional semantics.

Commonly, the similarity of two semantic vectors is computed by taking their cosine measure, which is the sum of the product of the basis weights of the vectors:

$$\text{cosine}(\vec{a}, \vec{b}) = \frac{\sum_i c_i^a c_i^b}{\sqrt{\sum_i (c_i^a)^2 \sum_i (c_i^b)^2}}$$

where c_i^a and c_i^b are the basis weights for \vec{a} and \vec{b} , respectively. However, other options may be a better fit for certain implementations, typically dependent on the weighting scheme. Some of these are surveyed by [21]. Throughout this thesis, I will use the cosine measure as the principal similarity metric for vector comparison because it is the most commonly used metric for the models I will consider and compare my work to.

To continue with our example, we wish to compute the relative similarity of ‘dog’ and

‘cat’ versus that of ‘dog’ and ‘snake’, to verify that dogs and cats are more similar in our model than dogs and snakes. We first compute:

$$\langle \vec{\text{dog}} | \vec{\text{cat}} \rangle = [2 \ 2 \ 1] \begin{bmatrix} 3 \\ 1 \\ 0 \end{bmatrix} = 2 \cdot 3 + 2 \cdot 1 + 1 \cdot 0 = 8$$

To obtain the cosine measure, we must normalise the inner product by the product of vector lengths

$$\|\vec{\text{dog}}\| \cdot \|\vec{\text{cat}}\| = \sqrt{2^2 + 2^2 + 1^2} \cdot \sqrt{3^2 + 1^2 + 0^2} = 3 \sqrt{10}$$

to obtain the cosine measure

$$\text{cosine}(\vec{\text{dog}}, \vec{\text{cat}}) = \frac{8}{3 \sqrt{10}} \approx 0.84$$

We apply the same procedure for ‘dog’ and ‘snake’, first calculating the inner product of the semantic vectors

$$\langle \vec{\text{dog}} | \vec{\text{snake}} \rangle = [2 \ 2 \ 1] \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix} = 2 \cdot 0 + 2 \cdot 2 + 1 \cdot 2 = 6$$

and then the normalisation coefficient

$$\|\vec{\text{dog}}\| \cdot \|\vec{\text{snake}}\| = \sqrt{2^2 + 2^2 + 1^2} \cdot \sqrt{0^2 + 2^2 + 2^2} = 3 \sqrt{8}$$

to obtain the cosine measure

$$\text{cosine}(\vec{\text{dog}}, \vec{\text{snake}}) = \frac{6}{3 \sqrt{8}} \approx 0.71$$

Interpreting the cosine measure as a percentage or degree of conceptual similarity, we see that ‘dog’ is more than 10% closer in meaning to ‘cat’ than it is to ‘snake’.

Readers interested in learning more about these aspects of distributional lexical semantics are invited to consult [21], which contains an extensive overview of implementation options for distributional models of word meaning.

2.3 Compositionality and Vector Space Models

In the above overview of distributional models of lexical semantics, we have seen that distributional semantic models (DSMs) are a rich and innovative way of learning word meaning from a corpus, and obtaining a measure of semantic similarity for words. However, it should be fairly obvious that the same method cannot be applied to sentences, whereby the meaning of a sentence would be given by the distribution of other sentences with which it occurs.

First and foremost, a sentence typically occurs only once in a corpus, and hence substantial and informative distributions cannot be created in this manner. More importantly, human ability to understand new sentences is a compositional mechanism: we understand sentences we have never seen before because we can generate sentence meaning from the words used, and how they are *put into relation*. To go from word vectors to sentence vectors, we must provide a *composition operation* allowing us to construct a sentence vector from a collection of word vectors. In this section, I will discuss several approaches to solving this problem, their advantages, and their limitations.

2.3.1 Additive Models

The simplest composition operation that comes to mind is straightforward vector addition, such that:

$$\vec{ab} = \vec{a} + \vec{b}$$

Conceptually speaking, if we view word vectors as semantic information distributed across a set of properties associated with basis vectors, using vector addition as a semantic composition operation states that the semantic information of a set of lemmas in a sentence is simply the sum of the semantic information of the individual lemmas. While crude, this approach is computationally cheap, and appears sufficient for certain NLP tasks: [55] shows it to be sufficient for automated essay marking tasks, and [33] shows it to perform better than a collection of other simple similarity metrics for summarisation, sentence paraphrase, and document paraphrase detection tasks.

However there are two principal objections to additive models of composition: first, vector addition is commutative, therefore $\overrightarrow{\text{John drank wine}} = \overrightarrow{\text{John}} + \overrightarrow{\text{drank}} + \overrightarrow{\text{wine}} = \overrightarrow{\text{Wine drank John}}$, and thus vector addition ignores syntactic structure completely; and second, vector addition sums the information contained in the vectors, effectively jumbling the meaning of words together as sentence length grows.

The first objection is problematic, as the syntactic insensitivity of additive models leads

them to equate the representations of sentences with patently different meanings. [63] propose to add some degree of syntactic sensitivity—namely accounting for word order—by weighting word vectors according to their order of appearance in a sentence as follows:

$$\vec{ab} = \alpha \vec{a} + \beta \vec{b}$$

where $\alpha, \beta \in \mathbb{R}$. Consequently $\overrightarrow{\text{John drank wine}} = \alpha \cdot \overrightarrow{\text{John}} + \beta \cdot \overrightarrow{\text{drank}} + \gamma \cdot \overrightarrow{\text{wine}}$ would not have the same representation as $\overrightarrow{\text{Wine drank John}} = \alpha \cdot \overrightarrow{\text{wine}} + \beta \cdot \overrightarrow{\text{drank}} + \gamma \cdot \overrightarrow{\text{John}}$.

The question of how to obtain weights and whether they are only used to reflect word order or can be extended to cover more subtle syntactic information is open, but it is not immediately clear how such weights may be obtained empirically and whether this mode of composition scales well with sentence length and increase in syntactic complexity. [42] suggests using machine-learning methods such as partial least squares regression to determine the weights empirically, but states that this approach enjoys little success beyond minor composition such as adjective-noun or noun-verb composition, and that there is a dearth of metrics by which to evaluate such machine learning-based systems, stunting their growth and development at the time of writing.

The second objection states that vector addition leads to increase in ambiguity as we construct sentences, rather than decrease in ambiguity as we would expect from giving words a context. For this reason, [63] suggest replacing additive models with multiplicative models as discussed in the next section, or combining them with multiplicative models to form mixture models as discussed in §2.3.3.

2.3.2 Multiplicative Models

The multiplicative model of [63] is an attempt to solve the ambiguity problem, discussed in the previous section, and provide implicit disambiguation during composition. The composition operation proposed is the component-wise multiplication (\odot) of two vectors. Vectors are expressed as the weighted superposition of their basis vectors, and the weights of the basis vectors of the composed vector is the product of the weights of the original vectors; for $\vec{a} = \sum_i c_i \vec{n}_i$, and $\vec{b} = \sum_i c'_i \vec{n}_i$, we have

$$\vec{ab} = \vec{a} \odot \vec{b} = \sum_i c_i c'_i \vec{n}_i$$

Such multiplicative models are shown by [63] to perform better at verb disambiguation tasks than additive models for noun-verb composition, against a baseline set by the original verb vectors. The experiment that they use to demonstrate this improvement will also serve

to evaluate our own models, and form the basis for further experiments, as discussed below in Chapter 6.

This approach to compositionality still suffers from two conceptual problems: first, component-wise multiplication remains commutative and hence word order is not accounted for; second, rather than ‘diluting’ information during large compositions and creating ambiguity, it may remove too much information through the ‘filtering’ effect of component-wise multiplication.

The first problem is more difficult to deal with for multiplicative models than for additive models, since both scalar multiplication and component-wise multiplication are commutative linear operations and hence $\alpha \vec{a} \odot \beta \vec{b} = \alpha \beta \vec{a} \odot \vec{b}$ and thus word order cannot be taken into account using scalar weights.

To illustrate how the second problem entails that multiplicative models do not scale well with sentence length, let us look at the structure of component-wise multiplication again: $\vec{a} \odot \vec{b} = \sum_i c_i c'_i \vec{n}_i$. For any i , if $c_i = 0$ or $c'_i = 0$, then $c_i c'_i = 0$, and therefore for any composition, the number of non-zero basis weights of the produced vector is less than or equal to the number of non-zero basis weights of the original vectors: at each composition step information is filtered out (or preserved, but never increased). Hence as the number of vectors to be composed grows, the number of non-zero basis weights of the product vector stays the same or—more realistically—decreases. Therefore for any composition of the form $\overrightarrow{a_1 \dots a_i \dots a_n} = \vec{a}_1 \odot \dots \odot \vec{a}_i \odot \dots \odot \vec{a}_n$, if there exist two subsets of the set of vectors involved such that the component-wise multiplication of the vectors in one subset forms a vector orthogonal to that formed from the component-wise multiplication of the vectors in the other, then $\overrightarrow{a_1 \dots a_i \dots a_n} = \vec{0}$. It follows that purely multiplicative models alone are not apt as a single mode of composition beyond binary composition operations.

One solution to this second problem not discussed by [63] would be to introduce some smoothing factor $s \in \mathbb{R}^+$ for point-wise multiplication such that $\vec{a} \odot \vec{b} = \sum_i (c_i + s)(c'_i + s)\vec{n}_i$, ensuring that information is never completely filtered out. Seeing how the problem of syntactic insensitivity still stands in the way of full-blown compositionality for multiplicative models, I leave it to those interested in salvaging purely multiplicative models to determine whether some suitable value of s can be determined.

2.3.3 Mixture Models

The problems faced by multiplicative models presented in §2.3.2 are acknowledged in passing by [63], who propose mixing additive and multiplicative models in the hope of leveraging the advantage of each while doing away with their pitfalls. This is simply expressed

as the weighted sum of additive and multiplicative models:

$$\vec{ab} = \alpha \vec{a} + \beta \vec{b} + \gamma(\vec{a} \odot \vec{b})$$

where α , β and γ are pre-determined scalar weights.

The problems for these models are threefold. First, the question of how scalar weights are to be obtained still needs to be addressed. Mitchell and Lapata [63] concede that one advantage of purely multiplicative models over weighted additive or mixture models is that the lack of scalar weights removes the need to optimise the scalar weights for particular tasks (at the cost of not accounting for syntactic structure), and avoids the methodological concerns accompanying this requirement.

Second, the question of how well this process scales from noun-verb composition to more syntactically rich expressions must be addressed. Using scalar weights to account for word order seems ad-hoc and superficial, as there is more to syntactic structure than the mere ordering of words. Therefore an account of how to build sentence vectors for sentences such as “The dog bit the man” and “The man was bitten by the dog” in order to give both sentences the same (or a similar) representation would need to give a richer role to scalar weights than just token order. Perhaps specific weights could be given to particular syntactic classes (such as nouns) to introduce a more complex syntactic element into vector composition; but it is clear that this alone is not a solution, as the weight for nouns “dog” and “man” would be the same, allowing for the same commutative degeneracy observed in non-weighted additive models, in which $\vec{\text{the dog bit the man}} = \vec{\text{the man bit the dog}}$. Introducing a mixture of weighting systems accounting for both word order and syntactic roles may be a solution; however, it is not only ad-hoc but also arguably only partially reflects the syntactic structure of the sentence.

The third problem is that [63] show that in practice, while mixture models perform better at verb disambiguation tasks than additive models and weighted additive models, they perform equivalently to purely multiplicative models with the added burden of requiring parametric optimisation of the scalar weights.

Therefore while mixture models aim to take the best of additive and multiplicative models while avoiding their problems, they are only partly successful in achieving the latter goal, and demonstrably not much better in achieving the former.

2.3.4 Tensor-Based Models

From §§2.3.1–2.3.3 we observe that the need for incorporating syntactic information into DSMs to achieve true compositionality is pressing, if only to develop a non-commutative

composition operation that can take into account word order without the need for ad-hoc weighting schemes, and hopefully to integrate richer syntactic information as well.

An early proposal by Smolensky [74, 75] to use linear algebraic tensors as a composition operation solves the problem of finding non-commutative vector composition operators. The composition of two vectors is their tensor product, sometimes called the Kronecker product when applied to vectors rather than vector spaces: for $\vec{a} \in V = \sum_i c_i \vec{n}_i$, and $\vec{b} \in W = \sum_j c'_j \vec{n}'_j$, we have:

$$\vec{ab} = \vec{a} \otimes \vec{b} = \sum_{ij} c_i c'_j \vec{n}_i \otimes \vec{n}'_j$$

To illustrate with an example consider the following two vectors:

$$\vec{a} = [a_1 \ a_2]^\top \quad \vec{b} = [b_1 \ b_2]^\top$$

Their Kronecker product is as follows:

$$\vec{a} \otimes \vec{b} = \begin{bmatrix} a_1 b_1 & a_1 b_2 \\ a_2 b_1 & a_2 b_2 \end{bmatrix}$$

The composition operation takes the original vectors and maps them to a vector in a larger vector space $V \otimes W$ which is the tensor space of the original vectors' spaces. Here the second instance of \otimes is not a recursive application of the Kronecker product, but rather the pairing of basis elements of V and W to form a basis element of $V \otimes W$. The shared notation and occasional conflation of Kronecker and tensor products may seem confusing, but is fairly standard in multi-linear algebra.

The advantage of this approach is twofold. First, vectors for different words need not live in the same spaces but can be composed nonetheless. This allows us to represent vectors for different word classes (e.g. topics, syntactic roles, etc.) in different spaces with different bases, which was not possible under additive or multiplicative models. Second, because the product vector lives in a larger space, we obtain the intuitive notion that the information of the whole is richer and more complex than the mere sum or product of the information of the parts.

2.3.4.1 Dimensionality Problems

However this increase in dimensionality brings two rather large problems for tensor based models. The first is computational: the size of the product vector space is the product

of the size of the original vector spaces. If we assume that all words live in the same space N of dimensionality $\dim(N)$ then the dimensionality of an n -word sentence vector is $\dim(N)^n$. If we have as many basis vectors for our word semantic space as there are lexemes in our vocabulary—e.g. approximately 170k in English¹—then the size of our sentence vectors quickly reaches magnitudes for which vector comparison (or even storage) are computationally intractable². Even if, as most DSM implementations do, we restrict the basis vectors of word semantic spaces to the k (e.g. $k = 2000$) most frequent words in a corpus, the sentence vector size still grows exponentially with sentence length, and the implementation problems remain.

The second problem is mathematical: sentences of different length live in different vector spaces, and if we assign different vector spaces to different word types (e.g. syntactic classes), then sentences of different syntactic structure live in different vector spaces. As a result, they cannot be compared directly using inner product or cosine measure, leaving us with no obvious mode of semantic comparison for sentence vectors. If any model wishes to use tensor products in composition operations, it must find some way of reducing the dimensionality of product vectors to some common vector space so that they may be directly compared.

One notable method by which these dimensionality problems can be solved in general are the holographic reduced representations proposed by [67]. The product vector of two vectors is projected into a space of smaller dimensionality by circular convolution to produce a trace vector. The circular correlation of the trace and one of the original vectors produces a noisy version of the other original vector. The noisy vector can be used to recover the clean original vector by comparing it with a pre-defined set of candidates (for example the set of word vectors if our original vectors are word meanings). Traces can be summed to form new traces effectively containing several vector pairs from which original vectors can be recovered. Using this encoding/decoding mechanism, the tensor product of sets of vectors can be encoded in a space of smaller dimensionality, and then recovered for computation without ever having to fully represent or store the full tensor product, as discussed by [87].

There are problems with this approach that make it unsuitable for our purposes. First, there is a limit to the information that can be stored in traces, which is independent of the size of the vectors stored, but is a logarithmic function of their number. As we wish to be able to store information for sentences of variable word length without having to directly

¹Source: <http://www.oxforddictionaries.com/page/howmanywords>

²At four bytes per integer, and one integer per basis vector weight, the vector for “John loves Mary” would require roughly $(170000 \cdot 4)^3 \approx 280$ petabytes of storage, which is over ten times the data Google processes on a daily basis according to [23].

represent the tensored sentence vector, setting an upper bound to the number of vectors that can be composed in this manner limits the length of the sentences we can represent compositionally using this method.

Second, and perhaps more importantly, there are restrictions on the nature of the vectors that can be encoded in such a way: the vectors must be independently distributed such that the mean euclidean length of each vector is equal to one. Such conditions are unlikely to be met in word semantic vectors obtained from a corpus, and as the failure to do so affects the system’s ability to recover clean vectors, holographic reduced representations are not *prima facie* useable for compositional DSMs. However, it is important to note that [87] considers possible linguistic application areas where they may be of use, although once again these mostly involve noun-verb and adjective-noun compositionality rather than full-blown sentence vector construction. We retain from [67] the importance of finding methods by which to project the tensored sentence vectors into a common space for direct comparison, as will be discussed further in Chapter 3.

2.3.4.2 Syntactic Expressivity

An additional problem of a more conceptual nature is that using the tensor product as a composition operation simply preserves word order. As I discussed in §2.3.3, this is not enough on its own to model sentence meaning. We need to have some means by which to incorporate syntactic analysis into composition operations.

Early work on including syntactic sensitivity into DSMs by [40] suggests using syntactic dependency relations to determine the frame in which the distributions for word vectors are collected from the corpus, thereby embedding syntactic information into the word vectors. This idea is built upon by [14] who suggest incorporating dependency relations into tensor-based composition operations as vectors themselves. For example in the sentence “Simon loves red wine”, “Simon” is the subject of “loves”, “wine” is its object, and “red” is an adjective describing “wine”. Hence from the dependency tree with “loves” as root node, its subject and object as children, and their adjectival descriptors (if any) as their children, we read the following structure: $\overrightarrow{\text{loves}} \otimes \overrightarrow{\text{subj}} \otimes \overrightarrow{\text{Simon}} \otimes \overrightarrow{\text{obj}} \otimes \overrightarrow{\text{wine}} \otimes \overrightarrow{\text{adj}} \otimes \overrightarrow{\text{red}}$. Using the equality relation for inner products of tensor products:

$$\langle \vec{a} \otimes \vec{b} \mid \vec{c} \otimes \vec{d} \rangle = \langle \vec{a} \mid \vec{c} \rangle \times \langle \vec{b} \mid \vec{d} \rangle$$

we can therefore express inner-products of sentence vectors efficiently without ever having

to actually represent the tensored sentence vector:

$$\begin{aligned}
& \langle \overrightarrow{\text{Simon loves red wine}} \mid \overrightarrow{\text{Mary likes delicious rosé}} \rangle \\
&= \langle \overrightarrow{\text{loves}} \otimes \overrightarrow{\text{subj}} \otimes \overrightarrow{\text{Simon}} \otimes \overrightarrow{\text{obj}} \otimes \overrightarrow{\text{wine}} \otimes \overrightarrow{\text{adj}} \otimes \overrightarrow{\text{red}} \mid \overrightarrow{\text{likes}} \otimes \overrightarrow{\text{subj}} \otimes \overrightarrow{\text{Mary}} \otimes \overrightarrow{\text{obj}} \otimes \overrightarrow{\text{rosé}} \otimes \overrightarrow{\text{adj}} \otimes \overrightarrow{\text{delicious}} \rangle \\
&= \langle \overrightarrow{\text{loves}} \mid \overrightarrow{\text{likes}} \rangle \times \langle \overrightarrow{\text{subj}} \mid \overrightarrow{\text{subj}} \rangle \times \langle \overrightarrow{\text{Simon}} \mid \overrightarrow{\text{Mary}} \rangle \times \langle \overrightarrow{\text{obj}} \mid \overrightarrow{\text{obj}} \rangle \times \langle \overrightarrow{\text{wine}} \mid \overrightarrow{\text{rosé}} \rangle \times \langle \overrightarrow{\text{adj}} \mid \overrightarrow{\text{adj}} \rangle \times \langle \overrightarrow{\text{red}} \mid \overrightarrow{\text{delicious}} \rangle \\
&= \langle \overrightarrow{\text{loves}} \mid \overrightarrow{\text{likes}} \rangle \times \langle \overrightarrow{\text{Simon}} \mid \overrightarrow{\text{Mary}} \rangle \times \langle \overrightarrow{\text{wine}} \mid \overrightarrow{\text{rosé}} \rangle \times \langle \overrightarrow{\text{red}} \mid \overrightarrow{\text{delicious}} \rangle
\end{aligned}$$

This example shows that this formalism allows for the comparison of sentences with identical dependency trees to be broken down to term-to-term comparison without the need for the tensor products to ever be computed or stored, reducing computation to inner product calculations.

However while matching up terms with identical syntactic roles in the sentence works well in the above example, this model suffers from the same problems as the original tensor-based compositionality of [74] in that, by the authors’ own admission, sentences of different syntactic structure live in spaces of different dimensionality and thus cannot be directly compared. Hence we cannot use this to measure the similarity between even small variations in sentence structure, such as the pair “Simon likes red wine” and “Simon likes wine”, which are sentences of different length and grammatical structure, and therefore live in different vector spaces under this approach.

2.3.5 SVS Models

The idea of including syntactic relations to other lemmas in word representations discussed in §2.3.4, above, is applied differently in the structured vector space model presented by [26]. They propose to represent word meanings not as simple vectors, but as triplets:

$$w = (v, R, R^{-1})$$

where v is the word vector, constructed as in any other DSM, R and R^{-1} are selectional preferences, and take the form of $\mathcal{R} \rightarrow \mathcal{D}$ maps where \mathcal{R} is the set of dependency relations and \mathcal{D} is the set of word vectors. Selectional preferences are used to encode the lemmas that w is typically the parent of in the dependency trees of the corpus in the case of R , and typically the child of in the case of R^{-1} .

Composition takes the form of vector updates according to the following protocol. Let $a = (v_a, R_a, R_a^{-1})$ and $b = (v_b, R_b, R_b^{-1})$ be two words being composed, and let r be the

dependency relation linking a to b . The vector update procedure is as follows:

$$\begin{aligned} a' &= (v_a \odot R_b^{-1}(r), R_a - \{r\}, R_a^{-1}) \\ b' &= (v_b \odot R_a(r), R_b, R_b^{-1} - \{r\}) \end{aligned}$$

where a', b' are the updated word meanings, and \odot is whichever vector composition (addition, component-wise multiplication) we wish to use. The word vectors in the triplets are effectively filtered by combination with the lemma which the word they are being composed with expects to bear relation r to, and this relation between the composed words a and b is considered to be used and hence removed from the domain of the selectional preference functions used in composition.

This mechanism is therefore a more sophisticated version of the compositional disambiguation mechanism discussed by [63] in that the combination of words filters the meaning of the original vectors which may be ambiguous (e.g. if we have one vector for all senses of the word “bank”); however, contrary to [63] the information of the original vectors is modified but essentially preserved, allowing for further combination with other terms, rather than directly producing a joint vector for the composed words. The added fact that R and R^{-1} are partial functions associated with specific lemmas forces grammaticality during composition, since if a holds a dependency relation r to b which it never expects to hold (for example a verb having as subject another verb, rather than the reverse) then R_a and R_b^{-1} are undefined for r and the update fails. However, there are some problems with this approach if our goal is true compositionality.

First, this model does not allow some of the ‘repeated compositionality’ we need because of the update of R and R^{-1} . For example, we expect that an adjective composed with a noun produces something *like* a noun in order to be further composed with a verb or even another adjective. However here, because the relation *adj* would be removed from R_b^{-1} for some noun b composed with an adjective a , this new representation b' would not have the properties of a noun in that it would no longer expect composition with an adjective, rendering representations of simple expressions like “the new red car” impossible. Of course, we could remove the update of the selectional preference functions from the compositional mechanism, but then we would lose this attractive feature of grammaticality enforcement through the partial functionality of R and R^{-1} .

Second, this model does little more than represent the implicit disambiguation which is expected during composition, rather than actually provide a full blown compositional model. The inability of this system to provide a novel mechanism by which to obtain a joint vector for two composed lemmas—thereby building towards sentence vectors—

entails that this system provides no means by which to obtain semantic representations of larger syntactic structures which can be compared by inner product or cosine measure as is done with any other DSM. Of course, this model could be combined with the compositional models presented in §§2.3.1–2.3.3 to produce sentence vectors, but while some syntactic sensitivity would have been obtained, the word ordering and other problems of the aforementioned models would still remain, and little progress would have been made towards true compositionality.

Finally, [26] state that the selectional preferences for each word must be pre-defined, but provide no procedure by which we would learn these from a corpus. This is an open problem for this model rather than a conceptual objection, but it remains to be seen if such preferences can be empirically determined without affecting the performance or tractability of such a model.

We retain from this attempt to introduce compositionality in DSMs that including information obtained from syntactic dependency relations is important for proper disambiguation. We also note that having some mechanism by which the grammaticality of the expression being composed is a pre-condition for its composition is a desirable feature for any compositional mechanism.

2.4 Matrix-Based Compositionality

The final class of approaches to vector composition I wish to discuss are three matrix based models.

2.4.1 Generic Additive Model

The first is the Generic Additive Model of [90]. This is a generalisation of the weighted additive model presented in §2.3.1. In this model, lexical vectors are not just multiplied by fixed parameters α and β before adding them to form the representation of their combination. Instead, they are the arguments of matrix multiplication by square matrices A and B as follows:

$$\vec{ab} = A\vec{a} + B\vec{b}$$

Here A and B represent the added information provided by putting two words into relation.

The numerical content of A and B is learned by linear regression over triplets $(\vec{a}, \vec{b}, \vec{c})$ where \vec{a} and \vec{b} are lexical semantic vectors, and \vec{c} is the *expected* output of the combination of \vec{a} and \vec{b} . This learning system thereby requires the provision of labelled data for

linear regression to be performed. Zanzotto et al. suggest several sources for this labelled data, such as dictionary definitions and word etymologies.

This approach is richer than the weighted additive models, since the matrices act as linear maps on the vectors they take as ‘arguments’, and thus can encode more subtle syntactic or semantic relations. However, this model treats all word combinations as the same operation, e.g. treating the combination of an adjective with its argument and a verb with its subject as the same sort of composition. This may not be fundamentally wrong, but it would be worthwhile evaluating how well this system works in the context of experiments such as those presented in Chapter 6. But because of the diverse ways there are of training such supervised models, we leave it to those who wish to further develop this specific line of research to perform such evaluations.

2.4.2 Adjective Matrices

The second approach is the matrix-composition model of [5], which they develop only for the case of adjective-noun composition, although their approach can seamlessly be used for any other predicate-argument composition. Contrary to most of the approaches above, which aim to combine two lexical vectors to form a lexical vector for their combination, Baroni and Zamparelli suggest giving different semantic representations to different types, or more specifically to adjectives and nouns.

In this model, nouns are lexical vectors, as with other models. However, embracing a view of adjectives that is more in line with formal semantics than with distributional semantics, they model adjectives as linear maps taking lexical vectors as input and producing lexical vectors as output. Such linear maps can be encoded as square matrices, and applied to their arguments by matrix multiplication. Concretely, let $M^{adjective}$ be the matrix encoding the adjective’s linear map, and \overrightarrow{noun} be the lexical semantic vector for a noun; their combination is simply

$$\overrightarrow{adjective\ noun} = M^{adjective} \times \overrightarrow{noun}$$

Similarly to the Generic Additive Model described above, the matrix for each adjective is learned by linear regression over a set of pairs $(\overrightarrow{noun}, \overrightarrow{c})$ where the vectors \overrightarrow{noun} are the lexical semantic vectors for the arguments of the adjective in a corpus, and \overrightarrow{c} is the semantic vector corresponding to the expected output of the composition of the adjective with that noun.

This may, at first blush, also appear to be a supervised training method for learning adjective matrices from ‘labelled data’, seeing how the expected output vectors are needed. However, Baroni and Zamparelli work around this constraint by automatically producing

the labelled data from the corpus by treating the adjective-noun compound as a single token, and learning its vector using the same distributional learning methods they used to learn the vectors for nouns.

This same approach can be extended to other unary relations without change. It bears some similarity to some of the work presented in this document, and a direct comparison of our frameworks would be interesting. However, this requires a method of extending this approach to binary predicates for full comparison to be made. Furthermore, this approach has a larger set of free parameters, such as the method of dimensionality reduction used, the linear regression learning algorithm and its parameters, and so on. While this entails that this class of models can certainly be tweaked to improve results, the difficult task of finding optimal parameters for such a framework is outside of the scope of this thesis, and we will not attempt in our evaluations of Chapter 6 an unfair direct comparison by using substandard parameters.

2.4.3 Recursive Matrix-Vector Model

The third approach is the recently-developed Recursive Matrix-Vector Model (MV-RNN) of [76]³, which claims the two matrix-based models described above as special cases. In MV-RNN, words are represented as a pairing of a lexical semantic vector \vec{a} with an operation matrix A . Within this model, given the parse of a sentence in the form of a binarised tree, the semantic representation (\vec{c}, C) of each branch node in the tree is produced by performing the following two operations on its children (\vec{a}, A) and (\vec{b}, B) .

First, the vector component \vec{c} is produced by applying the operation matrix of one child to the vector of the other, and vice versa, and then projecting the concatenation of both of the products back into the same vector space as the child vectors using a projection matrix W , which must also be learned:

$$\vec{c} = g \left(W \times \begin{bmatrix} B \times \vec{a} \\ A \times \vec{b} \end{bmatrix} \right)$$

Here, g is a component-wise activation function to be determined by the model implementation (e.g. a sigmoid function), used to introduce an element of non-linearity to the compositional process.

Second, the matrix C is calculated by projecting the pairing of matrices A and B back

³Other similar related deep-learning approaches are also worth investigating, such as that of [46].

into the same space, using a projection matrix W_M , which must also be learned:

$$C = W_M \times \begin{bmatrix} A \\ B \end{bmatrix}$$

The pairing (\vec{c}, C) obtained through these operations forms the semantic representation of the phrase falling under the scope of the segment of the parse tree below that node.

This approach to compositionality yields good results in the experiments described in [76]. It furthermore has appealing characteristics such as treating relational words differently through their operation matrices, and allowing for recursive composition, as the output of each composition operation is of the same type of object as its inputs. However, the learning procedure for the projection matrices and the operation matrices for each word, described in [76], is non-trivial. It relies on a supervised learning algorithm, and thus is open to the same problems of data availability and sparsity as any other supervised learning system. This also makes it difficult to produce a version of the model to run the experiments described in Chapter 6, although working towards joint evaluation of the models described in this section with the MV-RNN approach will certainly need to be the object of further work.

The other significant difference with the compositional framework presented in this thesis is that composition in MV-RNN is always a binary operation, e.g. to compose a transitive verb with its subject and object one would first need to compose it with its object, and then compose the output of that operation with the subject. The framework discussed in this thesis allows for the construction of representations for relations of larger arities, permitting the simultaneous composition of a verb with its subject and object. Whether or not this theoretical difference leads to significant differences in composition quality requires joint evaluation which, once again, I will leave to further work.

Chapter 3

Foundations of DisCoCat

Chapter Abstract

This chapter describes the DisCoCat framework of [19], which forms the basis for most of the work presented in this thesis. It introduces the concepts of pregroup grammars, categories, diagrammatic calculi for categories, and of how information may be shared between mathematical formalisms through the medium of categories.

In Chapter 2, I discussed distributional semantic models (DSMs) and attempts to provide a vector composition operation over word meanings to form distributional sentence representations. In this chapter, I will present an existing formalism aiming to solve this compositionality problem, as well as the mathematical background required to understand it and further extensions, building on the features and failures of previously discussed attempts at syntactically-sensitive compositionality.

[13, 19] propose to adapt a category theoretic model initially used to describe information flow in quantum information theory to the task of composing of semantic vectors. Syntactic analysis in the form of pregroup grammars—a type of categorial grammar—is given categorical semantics in order to be represented as a compact closed category P (a concept explained below), the objects of which are syntactic types and the morphisms of which are the reductions forming the basis of syntactic analysis. Vectors for words reside in vector spaces containing semantic vectors for lemmas of a particular syntactic type, and the set of vector spaces is represented as a compact closed category \mathbf{FVect} with vector spaces as objects and linear maps as morphisms.

The key feature of category theory exploited here is the ability to express different mathematical formalisms as structures which can be related, even if the original formalisms belong in different branches of mathematics. Hence the product category $P \times \mathbf{FVect}$ allows us

to relate syntactic types to vector spaces and syntactic reductions to linear maps so that we obtain a mechanism by which *syntactic analysis guides semantic composition operations*.

This pairing of syntactic analysis and semantic composition ensures both that grammaticality restrictions are in place as in the model of [26], and that syntactically-driven semantic composition in the form of inner-products provides the implicit disambiguation features as in the compositional models of [26] and [63]. The composition mechanism also involves projection of tensored vectors into a common semantic space without the need for computing the full representation of the tensored vectors (in a manner similar to [67]), but without the added restriction as to the nature of the vector spaces it can be applied to. This avoids the complexity and comparison problems faced by other tensor-based composition mechanisms such as those of [74] and [14].

The word vectors can be specified model-theoretically and the sentence space can be defined over boolean values to obtain grammatically-driven truth-theoretic semantics in the style of [65], as proposed by [13]. Some logical operators can be emulated in this setting, such as using swap matrices for negation as shown by [19]. Alternatively, corpus-based variations on this formalism have been proposed by [39] to obtain a non-truth theoretic semantic model of sentence meaning for which logical operations have yet to be defined.

Before explaining how this formalism works, in §3.3, I will introduce pregroup grammars in §3.1, and the required basics of category theory in §3.2.

3.1 Pregroup Grammars

Presented by Lambek in [52, 53] as a successor to his non-commutative type-logical calculus presented in [51], pregroup grammars are a class of categorial grammars with pregroup semantics. They comprise atomic grammatical types which can combine to form compound types. A series of application rules allow for type-reductions, forming the basis of syntactic analysis. The pregroup semantics of this syntactic formalism are what interest us, as will be discussed in §3.3. However, our first step will be to show how this syntactic analysis formalism works, which will in turn require an introduction to pregroups.

3.1.1 Pregroups

A pregroup is an algebraic structure of the form $(P, \leq, \cdot, 1, (-)^l, (-)^r)$. Let us explain these elements individually:

- P is simply a set of objects $\{a, b, c, \dots\}$.

- \leq is a partial ordering relation on P .
- \cdot is an associative, non-commutative monoid multiplication operator, and can be conceived of as a function $\cdot : P \times P \rightarrow P$ such that if $a, b \in P$ then $a \cdot b \in P$. Therefore P is closed under this operation.
- $1 \in P$ is the unit, satisfying $a \cdot 1 = a = 1 \cdot a$ for all $a \in P$.
- $(-)^l$ and $(-)^r$ are the left and right adjoints, and can be conceived of as functions $(-)^l : P \rightarrow P$ and $(-)^r : P \rightarrow P$ such that for any $a \in P$, $a^l, a^r \in P$. Adjoints are further described by the following axioms:
 - Reversal: if $a \leq b$ then $b^l \leq a^l$ (as for a^r, b^r).
 - Ordering: $a \cdot a^r \leq 1 \leq a^r \cdot a$ and $a^l \cdot a \leq 1 \leq a \cdot a^l$.
 - Cancellation: $a^{lr} = a = a^{rl}$.
 - Equality of identity: $1^r = 1 = 1^l$.
 - Self-adjoint multiplication: $(a \cdot b)^r = b^r \cdot a^r$.

I say that a pregroup is freely generated by some basic set of types $\{a, b, c, \dots, z\}$ to mean that all elements of the pregroup, such as the adjoints $\{a^l, b^r, \dots\}$ and complex types $\{a \cdot b, c \cdot d \cdot e, c^r \cdot a \cdot b^l\}$ are formed by applying the adjoint operations $(-)^r$ and $(-)^l$ and the multiplication operation \cdot to elements of the basic set or those thus-generated from it. Notationally, this means that the only alphabet used in complex types is that used to enumerate objects of the basic set.

As a notational simplification I write ab for $a \cdot b$, and if $abcd \leq cd$ I write $abcd \rightarrow cd$ and call this a reduction, omitting the identity wherever it might appear. Monoid multiplication is associative, so parentheses may be added or removed for notational clarity without changing the meaning of the expression as long as they are not directly under the scope of an adjoint operator.

An example reduction in pregroup might be:

$$aa^rbc^lc \rightarrow bc^lc \rightarrow b$$

I note here that the reduction order is not always unique, as I could have reduced the expression as follows: $aa^rbc^lc \rightarrow aa^rb \rightarrow b$. As a further notational simplification, if there exists a chain of reductions $a \rightarrow \dots \rightarrow b$ we may simply write $a \rightarrow b$ (in virtue of the transitivity of partial ordering relations). Hence in our above example, we can express both reduction paths as $aa^rbc^lc \rightarrow b$.

3.1.2 Pregroups and Syntactic Analysis

Pregroups can be used for grammatical analysis by freely generating the set P of a pregroup from the combination of basic syntactic types n, s, \dots and defining one type (s) to be the sentence type. As in any categorial grammar, words of the lexicon are assigned one or more possible types (corresponding to different syntactic roles) in a pre-defined *type dictionary*, and the grammaticality of an expression is verified by demonstrating the existence of a reduction from the type of the expression to the sentence type s .

For example, let us assign to nouns the type n , and to transitive verbs the compound type $n^r sn^l$. We can read from the type of a transitive verb that it is something which ‘expects’ a noun on its left, and one on its right, in order to reduce to a sentence. A sample reduction of “John loves cake” with ‘John’ and ‘cake’ being nouns of type n and ‘loves’ being a verb of type $n^r sn^l$ is as follows:

$$n(n^r sn^l)n \rightarrow (sn^l)n \rightarrow s$$

And thus we see that the transitive verb has combined with the subject to become something that requires an object, which it obtains and then becomes a sentence. The expression reduces to s , and hence the expression is grammatical.

Intransitive verbs can be given the type $n^r s$ such that “John sleeps” would be analysed in terms of the reduction $n(n^r s) \rightarrow s$. Adjectives can be given the type nn^l such that “red round rubber ball” would be analysed by $(nn^l)(nn^l)(nn^l)n \rightarrow n$. And so on and so forth for other syntactic classes. . .

Lambek, in [53], presents the details of a slightly more complex pregroup grammar with a richer hierarchy of types than presented here. It is hand-constructed and iteratively extended by expanding the type hierarchy as previous versions of the grammar encounter unparseable expressions.

3.1.3 A graphical calculus for pregroups

Pregroups can be represented using a simple graphical calculus [69] allowing us to visually exhibit the simultaneous nature of type reductions in an elegant and intuitive manner. Cancellations of the type aa^r or $a^l a$ are represented as ‘cups’ as shown in Figure 3.1. I designate the non-reduction of a type by a single downward line, which can be seen as the ‘output’ of the reduction.

Figure 3.2 shows the diagrammatic reduction for the pregroup parse of “John loves Mary”, whereby a noun (“John”) of type n combines with the leftmost adjoint of the compound term for a transitive verb (“loves”) of type $n^r sn^l$, and another noun (“Mary”) com-

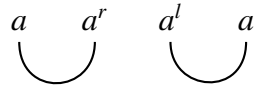


Figure 3.1: Diagrammatic examples of pregroup reductions.

bines with the rightmost adjoint of the verb to form a sentence type s .

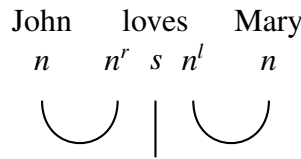


Figure 3.2: Diagrammatic pregroup parse of “John loves Mary”.

This diagrammatic calculus bears some striking similarities to the diagrammatic calculus for compact closed categories, described in §3.2.3. This similarity is no coincidence because of the relation between pregroups and compact closed categories, discussed in §3.2.2. This diagrammatic similarity will make it easier to visually describe the process of passing from syntactic analysis to semantic interpretation, as discussed in §3.3.

3.2 Categories

Category theory is a branch of pure mathematics which allows for the formulation of other mathematical structures and formalisms in terms of objects, arrows, and a few axioms. This simplicity and restricted conceptual language makes category theory both specific and general. It is specific in that the new properties of existing theories can be deduced from categorical axioms. It is general in that properties of these theories can be related to properties of other theories if they bear the same categorical representations.

It is this ability category theory provides to communicate information both within and across mathematical structures which makes it such a powerful tool. In this function, it has been at the centre of recent work in the foundations of physics and the modelling of quantum information flow, as presented in [1]. The connection¹ between the mathematics

¹I interpret this connection as one of loose analogy, at best, rather than holding the view that there is some fundamental link between quantum mechanics and language. It just happens that in both quantum mechanics and language, there is a notion of information being communicated, exchanged, or affected by objects with an “uncertain” state. In the case of quantum mechanics, this uncertainty takes the form of state superpositions, while in language it takes the form of ambiguity and polysemy. It should therefore come as

used for this branch of physics and those potentially useful for linguistic modelling has been noted by several sources, such as [86, 54, 83].

In this section, in order to demonstrate how these mathematical methods carry over to semantic analysis, I will briefly examine the basics of category theory, monoidal categories, and compact closed categories. The focus will be on defining enough basic concepts to proceed rather than provide a full-blown tutorial on category theory and the modelling of information flow, as several excellent sources already cover both aspects, e.g. [58, 85, 18]. A categories-in-a-nutshell crash course is also provided in [13, 19].

3.2.1 The Basics of Category Theory

Let us first consider the simplest definition of a category. A basic category \mathbf{C} is defined in terms of the following elements:

- A collection of objects $ob(\mathbf{C})$.
- A collection of morphisms $hom(\mathbf{C})$.
- A morphism composition operation \circ .

Each morphism f has a domain $dom(f) \in ob(\mathbf{C})$ and a codomain $codom(f) \in ob(\mathbf{C})$. For $dom(f) = A$ and $codom(f) = B$, I abbreviate these definitions as $f : A \rightarrow B$. Despite the notational similarity to function definitions, it is important to state that nothing else is pre-supposed about morphisms, and we should not treat them as functions.

The following axioms hold:

- For any $f : A \rightarrow B$ and $g : B \rightarrow C$ there exists $h : A \rightarrow C$ and $h = g \circ f$.
- For any $f : A \rightarrow B$, $g : B \rightarrow C$ and $h : C \rightarrow D$, \circ satisfies $(h \circ g) \circ f = h \circ (g \circ f)$.
- For every $A \in ob(\mathbf{C})$ there is an identity morphism $id_A : A \rightarrow A$ such that for any $f : A \rightarrow B$, $f \circ id_A = f = id_B \circ f$.

We can express various mathematical formalisms using such basic categories, and verify that these axioms hold. For example there is a category of sets with sets as objects and functions as morphisms, a category of posets with posets as objects and order-preserving

no surprise that the mathematics developed to deal with one of these domains might be adapted to deal with the other, but I believe that the connection stops there. If anything, the ability to straightforwardly adapt the mathematics of quantum information flow to linguistic information flow exemplifies the advantages provided by the abstractness and generality of category theory.

maps as morphisms, and a category of groups with groups as objects and group homomorphisms as morphisms, to name a few.

A product category $\mathbf{C} \times \mathbf{D}$ of two categories \mathbf{C} and \mathbf{D} is a category with pairs (A, B) as objects, where $A \in \text{ob}(\mathbf{C})$ and $B \in \text{ob}(\mathbf{D})$. There exists a morphism $(f, g) : (A, B) \rightarrow (C, D)$ in $\mathbf{C} \times \mathbf{D}$ if and only if there exists $f : A \rightarrow C \in \text{hom}(\mathbf{C})$ and $g : B \rightarrow D \in \text{hom}(\mathbf{D})$. Product categories are useful in attaining this desired generality of category theory, in that they allow us to relate objects and operations (morphisms) in one mathematical formalism or structure to those in another. However, this method of relating structures is not ideal, as will be discussed in Chapter 4, where a more elegant alternative will be provided (namely functors). For the time being, though, I will use product categories for the sake homogeneity with the work of [19], which this present work extends.

3.2.2 Compact Closed Categories

A slightly more complex class of categories is that of monoidal categories, which allow us to reason not just about objects and the relations between them, but also about combinations of objects in terms of the objects which they comprise. Formally, a (strict) monoidal category \mathbf{C} is a basic category to which we add a bifunctor \otimes (sometimes referred to as a *monoidal tensor*) satisfying the following conditions:

- For all $A, B \in \text{ob}(\mathbf{C})$ there is an object $A \otimes B \in \text{ob}(\mathbf{C})$.
- For all $A, B, C \in \text{ob}(\mathbf{C})$, we have $(A \otimes B) \otimes C \cong A \otimes (B \otimes C)$.
- There exists some $I \in \text{ob}(\mathbf{C})$ such that for any $A \in \text{ob}(\mathbf{C})$, we have $I \otimes A \cong A \cong A \otimes I$.
- For $f : A \rightarrow C$ and $g : B \rightarrow D$ in $\text{hom}(\mathbf{C})$ there is $f \otimes g : A \otimes B \rightarrow C \otimes D$ in $\text{hom}(\mathbf{C})$.
- For $f_1 : A \rightarrow C$, $f_2 : B \rightarrow D$, $g_1 : C \rightarrow E$ and $g_2 : D \rightarrow F$ the following equality holds:

$$(g_1 \otimes g_2) \circ (f_1 \otimes f_2) = (g_1 \circ f_1) \otimes (g_2 \circ f_2)$$

The strictness of the category entails that the isomorphisms described above are equalities.

A compact bi-closed category \mathbf{C} is a monoidal category with the following additional axioms:

- Each object $A \in \text{ob}(\mathbf{C})$ has left and right ‘adjoint’ objects A^l and A^r in $\text{ob}(\mathbf{C})$. The following isomorphism shows the distribution of adjoints over tensored objects:

$$(A \otimes B)^l \cong (B^l \otimes A^l) \quad \text{and} \quad (A \otimes B)^r \cong (B^r \otimes A^r)$$

- There exist four structural morphisms for each object $A \in \text{ob}(\mathbf{C})$:

$$- \eta_A^l : I \rightarrow A \otimes A^l.$$

$$- \eta_A^r : I \rightarrow A^r \otimes A.$$

$$- \epsilon_A^l : A^l \otimes A \rightarrow I.$$

$$- \epsilon_A^r : A \otimes A^r \rightarrow I.$$

- All such structural morphisms satisfy the following equalities:

$$- (1_A \otimes \epsilon_A^l) \circ (\eta_A^l \otimes 1_A) = 1_A.$$

$$- (\epsilon_A^r \otimes 1_A) \circ (1_A \otimes \eta_A^r) = 1_A.$$

$$- (1_{A^r} \otimes \epsilon_A^r) \circ (\eta^r \otimes 1_{A^r}) = 1_{A^r}.$$

$$- (\epsilon_A^l \otimes 1_{A^l}) \circ (1_{A^l} \otimes \eta_A^l) = 1_{A^l}.$$

Furthermore, for product categories involving compact closed categories, if there are pairings (a, A) and (b, B) , then there is a pairing $(a \otimes b, A \otimes B)$. One might describe compact closed categories as monoidal categories where we not only deal with the combination of objects, but also qualify how such combinations relate to simpler objects through ‘cancellations’ (ϵ morphisms) and ‘productions’ (η morphisms).

It is worth noting the obvious similarity between compact closed categories and the pregroup structures discussed in §3.1.1. I note that each object in a compact closed category has a left and a right adjoint, as do objects in pregroups. The monoidal tensor behaves identically to monoidal multiplication, and is also associative. There is a unit object I with the same equality properties as 1 in a pregroup. Furthermore, we note that if morphisms in a compact closed category are considered as ordering relations, the structural morphisms hold the same inequality relations as the object-adjoint pairings do in a pregroup.

We can therefore consider a pregroup as a compact closed category P modelling a poset. The elements of the pregroup’s set are the category’s objects; the ordering relations are its morphisms, 1 as I , and monoidal multiplication is the bifunctor \otimes . Notationally, instead of the single ordering relation symbol \leq we instead can write $\leq_{(a,b)}$ to denote the morphism expressing $a \leq b$. Likewise, the unary operators $(-)^l$ and $(-)^r$ can be turned into a set of morphisms linking types to their adjoints, where each morphism can be individually denoted $(-)_a^l$ for the case $(-)^l :: a \mapsto a^l$, and similarly $(-)_a^r$ for the case $(-)^r :: a \mapsto a^r$, for any such a in P .

The procedure described above is called giving categorical semantics to the pregroup. In §3.3, I will discuss how the little category theory we have seen here and this notion

of giving categorical semantics to other formalisms can aid us in achieving our goal of syntax-sensitive compositional DSMs.

3.2.3 A Graphical Calculus for Compact Closed Categories

Compact closed categories may appear to be very abstract mathematical entities to reason with and about. Fortunately, a graphical calculus, surveyed in [73], has been developed to provide both visual and practical support for these tasks. Proofs in this graphical calculus take the form of applications of diagrammatic rewrite rules which are sound and complete, and correspond to mathematical proofs about compact closed categories. This graphical calculus has basic elements and rewrite rules (and associated categorical meanings), some important ones of which are shown in Tables 3.1–3.3.

In Table 3.1, the basic elements are shown. I depict the flow of information as going from the top of the diagram towards the bottom, along the paths of wires typed with objects of the category. In some applications of this calculus, an opposite convention is used, whereby information flows from bottom to top.

Identity is seen as a naked wire. Morphisms are a box which transforms an input wire into an output wire of (possibly) different type. Morphism composition is two boxes on the same wire, while tensored morphisms are two wires side by side (with morphism boxes on them). A function over tensors is a box that take two wires in and outputs one or more wires. States (morphisms from the unit I to other elements of the category) are triangles with one or more output wires. Their co-state (morphisms from objects of the category to I) are represented as upside-down triangles with one or more input wires, where the star (*) usually stands for an adjoint, as shown in the ‘swing’ and ‘float’ rewrite rules, discussed below.

In Table 3.2, the diagrammatic forms for the structural morphisms of a compact closed category are shown. The “special” *structural morphisms* of a compact closed category have a specific representation in this diagrammatic calculus, instead of boxes. I represent the ϵ morphisms as ‘cups’ similar to those found in the diagrammatic pregroup calculus presented in §3.1.3, and the η morphisms as ‘caps’. Naturally, in similar diagrammatic calculi where the flow of information is from bottom to top, cups and caps stand for η and ϵ morphisms, respectively.

Finally, in Table 3.3, some of the key graphical re-write rules are shown, with names which are (for the most part) not “official”, but principally there to make it easier to talk about them. First the two “yank” rewrites show how the combination of a cup and a cap ‘cancel’ each other to produce an identity morphism, following the definitions of the struc-

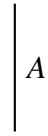
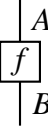
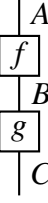
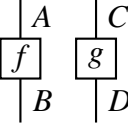
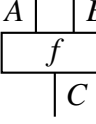
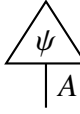
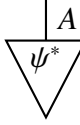
Name	Diagram	Meaning
Identity		$1_A : A \rightarrow A$
Morphism		$f : A \rightarrow B$
Composition		$g \circ f : A \rightarrow C$
Tensored functions		$f \otimes g : A \otimes B \rightarrow C \otimes D$
Morphisms over tensors		$h : A \otimes B \rightarrow C$
State		$\psi : I \rightarrow A$
Co-state		$\psi^* : A^* \rightarrow I$

Table 3.1: Basic elements of the graphical calculus for compact closed categories.

tural morphisms. The second shows how morphisms can “slide” up and down straight wires without changing the categorical meaning of the diagram. Both of these rewrite rules can be combined to show that morphisms can slide along non-straight wires (i.e. those including cups and caps) without changing the meaning of the diagram. This is exemplified in

Name	Diagram	Meaning
ϵ^r map		$\epsilon_A^r : A \otimes A^r \rightarrow I$
ϵ^l map		$\epsilon_A^l : A^l \otimes A \rightarrow I$
η^r map		$\eta_A^r : I \rightarrow A^r \otimes A$
η^l map		$\eta_A^l : I \rightarrow A \otimes A^l$

Table 3.2: Structural morphisms in the graphical calculus for compact closed categories.

Figure 3.3, which shows how a morphism can slide across such a non-straight wire. From left to right: I first use the yank equality, then the slide equality, then yank again to obtain the rightmost diagram from the leftmost. For those interested, this property is one of the central elements behind the diagrammatic proof of quantum teleportation [1, 16], the diagrammatic representation of which closely resembles that shown in Figure 3.3, with the inclusion of additional morphisms.

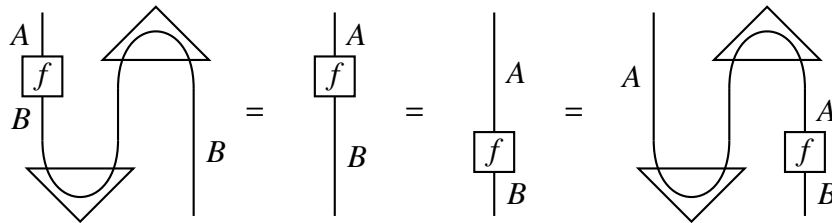


Figure 3.3: Examples of yank-slide equalities in the graphical calculus for compact closed categories.

Finally, the swing and float rules show that the yank operations can be separated out into separate steps, allowing us to ‘move’ states along cups and caps.

3.3 A Categorical Passage from Grammar to Semantics

In §3.2.2 I discussed how any pregroup grammar could be represented as a compact closed category P . In §3.2.1 I described how product categories allowed us to relate the objects and morphisms of one category to those of another. In this section, I will present how



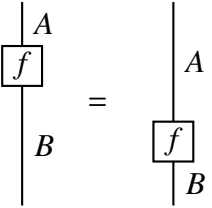
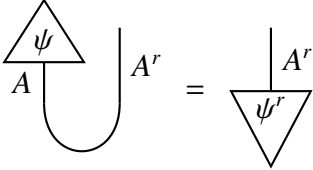
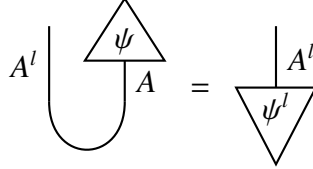
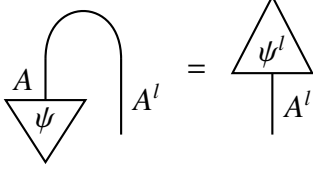
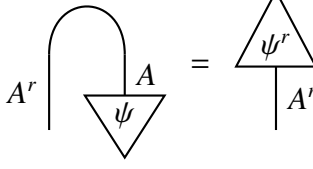
Name	Diagram	Meaning
Yank		$(1_A \otimes \epsilon_A^l) \circ (\eta_A^l \otimes 1_A) = 1_A$
Yank		$(\epsilon_A^r \otimes 1_A) \circ (1_A \otimes \eta_A^r) = 1_A$
Slide		$1_B \circ f = f \circ 1_A$
Swing		$\epsilon_A^r \circ (\psi \otimes 1_{A^r}) = \psi^l \circ 1_{A^r}$
Swing		$\epsilon_A^l \circ (1_{A^r} \otimes \psi) = \psi^r \circ 1_{A^l}$
Float		$(\phi \otimes 1_{A^l}) \circ \eta_A^l = 1_{A^l} \circ \psi^l$
Float		$(1_{A^r} \otimes \phi) \circ \eta_A^r = 1_{A^r} \circ \psi^r$

Table 3.3: Rewrite rules in the graphical calculus for compact closed categories.

[13, 19] suggest building on this by using categories to relate semantic composition to syntactic analysis in order to achieve syntax-sensitive composition in DSMs.

3.3.1 FVect

Let **FVect** be the symmetric monoidal compact closed category of finite-dimensional Hilbert spaces over \mathbb{R} , i.e. vector spaces over \mathbb{R} with orthogonal bases of finite dimension, and an inner product operation $\langle - | - \rangle : A \times A \rightarrow \mathbb{R}$ for every vector space A . The objects of **FVect** are the vector spaces, and the morphisms are linear maps between vector spaces. The unit object is \mathbb{R} and the monoidal tensor is the linear algebraic tensor product of vector spaces. The symmetric aspect of this category means that for any two objects $A \otimes B$ and $B \otimes A$ in the category, there exists an isomorphism $A \otimes B \cong B \otimes A$, corresponding here to the fact that any tensor is isomorphic to its permutations.

As a result of its symmetric nature, the category is degenerate in its adjoints, in that for any vector space A , we have the isomorphisms $A^l \cong A^r \cong A$. This is because the adjoint of a vector space A is its co-vector space $A^r = A^l = A^*$, the elements of which are the conjugate transposes of the vectors from that vector space. Since the conjugate transpose of a real-valued vector is just the transpose of that vector, each vector in some space A can be isomorphically mapped to a covector (its transpose) in A^* , hence $A \cong A^*$. As such, we can effectively do away with adjoints in this category, and ‘collapse’ ϵ^l , ϵ^r , η^l , and η^r maps into ‘adjoint-free’ ϵ and η maps. The structural morphisms of the category are the inner product operations ϵ ,

$$\epsilon_A : A \otimes A \rightarrow \mathbb{R} :: \vec{v} \otimes \vec{w} \mapsto \langle \vec{a} | \vec{b} \rangle$$

and the η maps from real numbers to tensored vector spaces

$$\eta_A : \mathbb{R} \rightarrow A \otimes A :: 1 \mapsto \overrightarrow{1_{A \otimes A}}$$

where $\overrightarrow{1_{A \otimes A}}$ is the superposition of all the basis vectors $\{\vec{a}_i \otimes \vec{a}_j\}_{ij}$ of $A \otimes A$

$$\overrightarrow{1_{A \otimes A}} = \sum_{ij} \vec{a}_i \otimes \vec{a}_j$$

On the diagrammatic front, I treat a vector $\vec{v} \in A$ as a state $\vec{v} : \mathbb{R} \rightarrow A$ and a co-vector as its co-state. This means that the application of ϵ maps to model the composition of vectors with a tensor corresponds to the application of the swing operation described above, showing how the vectors are brought into relation with the tensor through inner products, as shown in Figure 3.4.

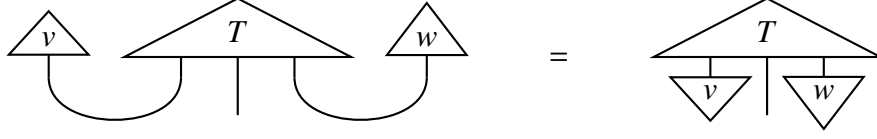


Figure 3.4: Sample diagrammatic representation of distributal composition.

3.3.2 Syntax Guides Semantics

If we consider the product category $P \times \mathbf{FVect}$, we observe that it has as objects pairs (a, A) where a is a pregroup type and A is a vector space, and as morphisms pairs (\leq, f) where \leq is a pregroup ordering relation and f is a linear map. By the definition of product categories, for any two vector space-type pairs (a, A) and (b, B) , there exists a morphism $(a, A) \rightarrow (b, B)$ only if there exists both an ordering $a \leq b$ and a linear map $A \rightarrow B$. If we view these pairings as the association of syntactic types with vector spaces containing semantic vectors for words of that type, this restriction effectively states that a linear map from A to B is only “permitted” in the product category if $a \leq b$.

Both P and \mathbf{FVect} being compact closed, it is simple to show that $P \times \mathbf{FVect}$ is as well, by considering the pairs of unit objects and structural morphisms from the separate categories: I is now $(1, \mathbb{R})$, and the structural morphisms are $(\epsilon_a^l, \epsilon_A)$, $(\epsilon_a^r, \epsilon_A)$, (η_a^l, η_A) , (η_a^r, η_A) . We are particularly interested in the ϵ maps, which are defined as follows (from the definition of product categories):

$$(\epsilon_a^l, \epsilon_A) : (a^l a, A \otimes A) \rightarrow (1, \mathbb{R}) \quad (\epsilon_a^r, \epsilon_A) : (a a^r, A \otimes A) \rightarrow (1, \mathbb{R})$$

This states that whenever there is a reduction step in the grammatical analysis of a sentence, there is a composition operation in the form of an inner product on the semantic front. Hence if nouns of type n live in some noun space N and transitive verbs of type $n^l s n^r$ live in some space $N \otimes S \otimes N$, then there must be some structural morphism of the form:

$$(\epsilon_n^r 1_s \epsilon_n^l, \epsilon_N \otimes 1_S \otimes \epsilon_N) : (n(n^r s n^l)n, N \otimes (N \otimes S \otimes N) \otimes N) \rightarrow (s, S)$$

We can read from this morphism the functions required to compose a sentence with a subject noun, a transitive verb, and an object noun, in order to obtain a vector living in some sentence space S , namely $(\epsilon_N \otimes 1_S \otimes \epsilon_N)$. Diagrammatically, this composition is represented as in Figure 3.4, where $\vec{v} \in N$ is the vector for the subject, $\vec{w} \in N$ is the vector for the object, and $T \in N \otimes S \otimes N$ is the tensor representing the noun.

The form of a syntactic type is therefore what dictates the structure of the semantic

space associated with it. The structural morphisms of the product category guarantee that for every syntactic reduction there is a semantic composition morphism provided by the product category: *syntactic analysis guides semantic composition*.

3.3.3 Example

To give an example, we can give syntactic type n to nouns, and $n^r s$ to intransitive verbs. The parse for “kittens sleep”, namely $nm^r s \rightarrow s$, corresponds to the morphism $\epsilon_n^r \otimes 1_s$ in P . The syntactic types dictate that the noun $\overrightarrow{\text{kittens}}$ lives in some vector space N , and the intransitive verb $\overrightarrow{\text{sleep}}$ in $N \otimes S$. The reduction morphism $(\epsilon_n^r 1_s)$ gives us the composition morphism $(\epsilon_N \otimes 1_S)$, which we can apply to $\overrightarrow{\text{kittens}} \otimes \overrightarrow{\text{sleep}}$.

Since we can express any vector as the weighted superposition of its basis vectors, let us expand $\overrightarrow{\text{kittens}} = \sum_i c_i^{\text{kittens}} \vec{n}_i$ and $\overrightarrow{\text{sleep}} = \sum_{ij} c_{ij}^{\text{sleep}} \vec{n}_i \otimes \vec{s}_j$. We can then express the composition as follows:

$$\begin{aligned}
\overrightarrow{\text{kittens sleep}} &= (\epsilon_N \otimes 1_S)(\overrightarrow{\text{kittens}} \otimes \overrightarrow{\text{sleep}}) \\
&= (\epsilon_N \otimes 1_S) \left(\sum_i c_i^{\text{kittens}} \vec{n}_i \otimes \sum_{jk} c_{jk}^{\text{sleep}} \vec{n}_j \otimes \vec{s}_k \right) \\
&= (\epsilon_N \otimes 1_S) \left(\sum_{ijk} c_i^{\text{kittens}} c_{jk}^{\text{sleep}} \vec{n}_i \otimes \vec{n}_j \otimes \vec{s}_k \right) \\
&= \sum_{ijk} c_i^{\text{kittens}} c_{jk}^{\text{sleep}} \langle \vec{n}_i | \vec{n}_j \rangle \vec{s}_k \\
&= \sum_{ik} c_i^{\text{kittens}} c_{ik}^{\text{sleep}} \vec{s}_k
\end{aligned}$$

The above equations are hopefully fairly clear at this stage: I express the vectors in their explicit form. I consolidate the sums by virtue of distributivity of the linear algebraic tensor product over addition; I then apply the tensored linear maps to the vector components (as the weights are scalars); and finally, I simplify the indices since $\langle \vec{n}_i | \vec{n}_j \rangle = 1$ if $\vec{n}_i = \vec{n}_j$ and 0 otherwise. I obtain a vector that lives in sentence space S .

Transitive sentences can be dealt with in a similar fashion:

$$\begin{aligned}
& \overrightarrow{\text{kittens chase mice}} \\
&= (\epsilon_N \otimes 1_S \otimes \epsilon_N)(\overrightarrow{\text{kittens}} \otimes \overrightarrow{\text{chase}} \otimes \overrightarrow{\text{mice}}) \\
&= (\epsilon_N \otimes 1_S \otimes \epsilon_N) \left(\sum_i c_i^{\text{kittens}} \overrightarrow{n_i} \otimes \left(\sum_{jkl} c_{jkl}^{\text{chase}} \overrightarrow{n_j} \otimes \overrightarrow{s_k} \otimes \overrightarrow{n_l} \right) \otimes \sum_m c_m^{\text{mice}} \overrightarrow{n_m} \right) \\
&= (\epsilon_N \otimes 1_S \otimes \epsilon_N) \left(\sum_{ijklm} c_i^{\text{kittens}} c_{jkl}^{\text{chase}} c_m^{\text{mice}} \overrightarrow{n_i} \otimes \overrightarrow{n_j} \otimes \overrightarrow{s_k} \otimes \overrightarrow{n_l} \otimes \overrightarrow{n_m} \right) \\
&= \sum_{ijklm} c_i^{\text{kittens}} c_{jkl}^{\text{chase}} c_m^{\text{mice}} \langle \overrightarrow{n_i} | \overrightarrow{n_j} \rangle \overrightarrow{s_k} \langle \overrightarrow{n_l} | \overrightarrow{n_m} \rangle \\
&= \sum_{ikm} c_i^{\text{kittens}} c_{ikm}^{\text{chase}} c_m^{\text{mice}} \overrightarrow{s_k}
\end{aligned}$$

In both cases, it is important to note that the tensor product passed as argument to the composition morphism, namely $\overrightarrow{\text{kittens}} \otimes \overrightarrow{\text{sleep}}$ in the intransitive case and $\overrightarrow{\text{kittens}} \otimes \overrightarrow{\text{chase}} \otimes \overrightarrow{\text{mice}}$ in the transitive case, never needs to be computed.

Part II

Theory

Chapter 4

Syntactic Extensions

Chapter Abstract

This chapter discusses a general methodology for the integration of syntactic formalisms into the DisCoCat framework by means of a functorial passage between the categorical representations of such formalisms and the category of vector spaces \mathbf{FVect} . It presents such categorical semantics and functorial passages for two grammatical formalisms: Context Free Grammars, and Lambek Grammars.

The DisCoCat framework presented in Chapter 3 makes use of pregroup grammars for syntactic analysis, principally because of their convenient algebraic structure which allows us to interpret any pregroup as a compact closed category. However, on a general level, the core strength of the framework is its ability to define the passage from syntax to semantics without commitment to the particular syntactic formalism used, or indeed the particular semantic mode of representation.

In this chapter, I will discuss how some other syntactic analysis formalisms can be substituted for the pregroup grammars of Chapter 3. In §4.1, I discuss the notion of a functor and of a functorial passage between categories, and demonstrate that the pair category $P \times \mathbf{FVect}$ used in §3.3.2 gives rise to a functor mapping syntactic analyses to semantic representations. In §4.2, I present a way in which generative grammars such as context free grammars may be supported in the framework. In §4.3, I will present work describing how such a functor can be defined for Lambek Grammar, how this requires the use of non-compact categories, and how an existing diagrammatic calculus can be used to reason about such categories.

4.1 Functorial Passages

In Chapter 3, a product category was used to model the passage from syntactic aspects of natural language composition to semantic representations of compositional operations and their arguments. There is, however, no requirement that we use a product category to model this passage. In fact—considering the compositional properties of functors described above—a functorial passage from syntax to semantics might fit our needs better. The idea of using a functorial passage from pregroup categories to **FVect** had been previously suggested by [68], and such a functorial passage has been described in a paper I co-authored with Coecke and Sardzadeh [17]. In this section, I will show that we can also define such a functorial passage from a product category to the category of vector spaces given certain restrictions, which are provided by the association made between pregroup types and vector spaces introduced in §3.3.2.

In §3.3.2 I discussed the construct developed by [13] of the pair category $P \times \mathbf{FVect}$ formed from a pregroup category P and the category of vector spaces **FVect**. We saw that this category was compact closed because P and **FVect** were, and that for any linear map $f : A \rightarrow B$ in **FVect** there are morphisms $(\leq, f) : x \otimes A \rightarrow y \otimes B$ for every x and y such that $x \leq y$. This definition actually covers a larger number of objects than we actually need, since it gives no explicit association between pregroup types and vector spaces. Yet this association is implicitly relied upon so that for every pregroup parse in P there is one and only one linear map in **FVect** corresponding to the composition operation associated with the parse. In other words, we rely on a pre-defined mapping $M : ob(P) \rightarrow ob(\mathbf{FVect})$ of pregroup types to vector spaces in order to be able to talk about *the* association between syntactic analysis and semantic composition, rather than *an* association between syntactic analysis and semantic composition. To make the implicit explicit: we say that a morphism $(\leq, f) : x \otimes A \rightarrow y \otimes B$ describes how f composes semantic vectors in vector space A standing for words of type x to produce a phrase vector in B for words of type y *if and only if* $M(x) = A$ and $M(y) = B$. This restriction effectively allows us to ignore all the other products in $ob(P \times \mathbf{FVect})$, and focus only on those pairs of products for which the morphisms between elements of the pair correspond to unique semantic interpretations of a pregroup parse.

To give a concrete example, consider the sentence “John loves Mary”, and suppose we give nouns the pregroup type n and transitive verbs the type $n^r sn^l$. We wish to model the semantic vectors for nouns in some vector space N and sentences of type s in some vector space S . Given the isomorphic relation between adjoints in **FVect**, we’d expect the vectors for transitive verbs to live in $N \otimes S \otimes N$, for the composition of nouns with the ‘argument

places' of the verb to even be possible. The parse of "John loves Mary" corresponds to the ordering $n(n^r sn^l)n \leq s$. This ordering corresponds to an arrow standing for the function $\epsilon_n^r \otimes 1_s \otimes \epsilon_n^l$ in P mapping the object $n(n^r sn^l)n$ to s . To find the morphism in \mathbf{FVect} allowing for the composition of the word vectors for 'John' in N with 'loves' in $N \otimes S \otimes N$ and 'Mary' in N to form a vector in the sentence space S , we look at the product category to find what maps $(n(n^r sn^l)n, N \otimes (N \otimes S \otimes N) \otimes N)$ to (s, S) . This morphism is, of course, one of the structural morphisms in $P \times \mathbf{FVect}$, namely $(\epsilon_n^r \otimes 1_s \otimes \epsilon_n^l, \epsilon_N \otimes 1_S \otimes \epsilon_N)$, as described in §3.3.2. However it should be noted that there are an infinite number of other pairings of vector spaces and linear maps with this parse which could have been considered, such as $(\epsilon_n^r \otimes 1_s \otimes \epsilon_n^l, \epsilon_N) : (n(n^r sn^l)n, N \otimes N) \rightarrow (s, 1)$ or $(\epsilon_n^r \otimes 1_s \otimes \epsilon_n^l, id_S) : (n(n^r sn^l)n, S) \rightarrow (s, S)$, amongst others, simply by virtue of the definition of product categories. These other objects and morphisms of $P \times \mathbf{FVect}$ are completely irrelevant to our task of associating syntax with semantics, yet they are there for us to consider. What allows us to discount them is the assumption, made earlier in this paragraph, that n should be associated with N , $n^r sn^l$ with $N \otimes S \otimes N$, and s with S , which is essentially an instance of the sort of mapping M discussed above. This shows us how the product-category approach works if such a mapping is defined, but that product categories come with a lot of extra material that is irrelevant, whereas the idea of defining a functor between P and \mathbf{FVect} has no such superfluous objects or morphisms and focuses on just the association of syntactic parses with semantic morphisms.

In this section, I begin by briefly describing functors in §4.1.1, before demonstrating in §4.1.2 that the mapping M between pregroup types and vector spaces and the product category $P \times \mathbf{FVect}$ can be used to canonically define a functor between P and \mathbf{FVect} .

4.1.1 Functors

The simplest way to think of a functor is as a map between two categories \mathbf{A} and \mathbf{B} which associates each object A in $ob(\mathbf{A})$ with some object $F(A)$ in $ob(\mathbf{B})$, and each morphism $f : A_1 \rightarrow A_2$ in $hom(\mathbf{A})$ with some morphism $F(f) : F(A_1) \rightarrow F(A_2)$ in $hom(\mathbf{B})$. Functors must furthermore preserve the structure of \mathbf{A} in \mathbf{B} by ensuring that the following two equations hold for all morphisms f and g of $hom(\mathbf{A})$ and all objects a of $ob(\mathbf{A})$:

1. $F(id_A) = id_{F(A)}$, i.e. if A maps to some $F(A)$ then its identity morphism maps to that same object's identity morphism.
2. $F(f \circ g) = F(f) \circ F(g)$, i.e. if two morphisms f and g compose to some $h = f \circ g$ in $hom(\mathbf{A})$, then there must be some $F(h)$ in $hom(\mathbf{B})$ which is the composition of $F(f)$ and $F(g)$.

Furthermore, a *strict* monoidal functor between monoidal categories is such that the following equality holds for two such categories \mathbf{A} and \mathbf{B} , equipped with monoidal tensors $\otimes_{\mathbf{A}}$ and $\otimes_{\mathbf{B}}$. For any objects A and B of $ob(\mathbf{A})$:

$$F(A \otimes_{\mathbf{A}} B) \cong F(A) \otimes_{\mathbf{B}} F(B)$$

To give a trivial example of a functor, consider the following two pregroups:

- P_1 , freely generated from the set $\{a, b\}$ with:
 - the ordering relation \leq^1 ,
 - the multiplicative operation \cdot_1 ,
 - the adjoint maps $(-^1)^l$ and $(-^1)^r$,
 - and the unit 1_1 .
- P_2 , freely generated from $\{c, d, e\}$, with:
 - the ordering relation \leq^2 ,
 - the multiplicative operation \cdot_2 ,
 - the adjoint maps $(-^2)^l$ and $(-^2)^r$,
 - the unit 1_2 ,
 - and the stipulation that $c \leq^2 d$.

We can define a functor F as follows:

- $F(1_1) = 1_2$, $F(a) = c$, $F(b) = d$ (by the definition of functors between monoidal categories, we therefore also have $F(ab) = cd$, $F(aab) = ccd$, and so on).
- $F((-^1)^l_i) = (-^2)^l_{F(i)}$ and $F((-^1)^r_i) = (-^2)^r_{F(i)}$ for any i in P_1 . E.g. $F((-^1)^l_a) = (-^2)^l_c$, so we map the adjoints of a in P_1 , such as a^l , to those of a 's "image" c in P_2 , such as c^l .
- $F(\leq^1_{(i,j)}) = \leq^2_{(F(i),F(j))}$ for any i, j in P_1 . E.g. $F(\leq^1_{(ba^l a, b)}) = \leq^2_{(dc^l c, d)}$, so effectively we map the reduction $ba^l a \rightarrow b$ in P_1 to $cd^l d \rightarrow d$ in P_2 .

The first condition for functors is satisfied by mapping identity morphisms of P_1 to those of P_2 . The second is satisfied primarily through the natural transitivity of ordering relations in both categories.

The important point to note here is that no type in P_1 is mapped by F to the type c of P_2 , or indeed to any compound types containing c (e.g. ac , bca , cc , etc.). Likewise,

even though F maps a to c and b to d , the absence of a morphism $\leq_{(a,b)}^1$ in P_1 means that there exist ordering morphisms in P_2 , such as $\leq_{(a,b)}^2$, which are *not* images of morphisms in P_1 even though the objects they connect are images of objects in P_1 . Functors can, in this sense, act like surjective mappings. They may, in this context, be seen as a way of explicitly embedding one mathematical structure into another (possibly larger) structure.

A functor is said to be ‘forgetful’ if it maps complex objects in one category to simpler objects in another category, where the simpler objects are “parts” of the more complex objects of the original category. A trivial example is a functor F between a category of pairs of atoms (of the form $(a, b), (b, d), \dots$) and a category of atoms (e.g. a, b, c, \dots) which maps each pair in the original category to the atom corresponding to the left element of the pair in the simpler category, and morphisms between pairs to morphisms between the left elements of the pair. For example F would map (a, b) and (a, c) to a , and some morphism $f : (a, b) \rightarrow (a, c)$ to the morphism $Ff : a \rightarrow a$. We can see here that F ‘forgets’ some of the information from the objects and morphisms of the original category by ignoring aspects of morphisms and objects relating to the right element of each pair.

Finally, functors can be interpreted as morphisms in the category of categories **Cat**, and hence the usual morphism composition rules apply for functors. Explicitly, for categories **A**, **B** and **C**, the functors $F : \mathbf{A} \rightarrow \mathbf{B}$ and $G : \mathbf{B} \rightarrow \mathbf{C}$ define a functor $H = G \circ F : \mathbf{A} \rightarrow \mathbf{C}$.

4.1.2 From Product Categories to Functors

We have just seen that functors can be composed like morphisms to define a functor from a category **A** to a category **C** through composition, if we are provided with ‘intermediate’ functors from **A** to **B** and **B** to **C**. This is an important property of functors within the context of our desire to link syntax to semantics. If a functor is defined between a pregroup category P and **FVect** (which, as will be discussed below, can be done for any pregroup category), then to use another grammatical formalism in lieu of pregroup grammars, it suffices to show that this other grammatical structure can be given categorical semantics as well, and that a functor can be defined from the relevant categorification of the grammar to some pregroup category. A functor from the new grammatical category to **FVect** then arises directly from composition. Putting this more formally: if it can be shown that there is a functor F mapping pregroup types in some pregroup category P to vector spaces in **FVect**, and reduction morphisms in P to linear maps in **FVect**, then if some grammatical formalism (e.g. a context free grammar, or combinatorial grammar, etc.) can be represented as a category P' for which there exists a functor $G : P' \rightarrow P$, it follows that the types and analysis morphisms of P' can be mapped directly to **FVect** by some functor $H = F \circ G$:

$P' \rightarrow \mathbf{FVect}$.

Let us then consider how we might define a functor using a mapping M between pregroup types and vector spaces, and the structure of the product category $P \times \mathbf{FVect}$. We begin by formally defining M as follows. Let \mathcal{T} be the set of basic pregroup types generating a pregroup P . This can be formalised by stating that any type t in P is written as

$$t : w_1^{m_1} \dots w_n^{m_n}$$

where $1 \leq n$, $w_i \in \mathcal{T}$ for all i , and m_i is an optional adjoint marker for w_i (e.g. l , ll , r , etc. , or no adjoint).

We associate, via M , a vector space A to each element a of the generating set \mathcal{T} , such that $M(a) = A$, without any commitments to the uniqueness of this association. This allows us to associate one vector space to several elements of the generating set, which might be desirable if, for example, we wish for nouns and noun phrases to have separate pregroup types (say n and \bar{n}), but for words/phrases of both types to reside in the same vector space so that we can compare the noun “dog” with the phrase “Mary’s favourite pet”.

We then recursively define the association of compound pregroup types with vector spaces as follows:

$$\begin{aligned} M(1) &= \mathbb{R} \\ M(ab) &= M(a) \otimes M(b) \\ M(a^r) &= M(a^l) = M(a)^* = A^* \end{aligned}$$

for any types a and b in $ob(P)$, and where $(V_1 \otimes \dots \otimes V_n)^* \cong (V_n \otimes \dots \otimes V_1)$. Because the set of pregroup types is freely generated, we can view the objects of a category $M(P)$ as being freely generated by a set $\{M(x) \mid x \in \mathcal{T}\}$, such that any object of the category $M(t)$ can be written as follows:

$$M(t) : M(w_1) \dots M(w_n) \quad \forall i. [w_i \in \mathcal{T}]$$

To give an example, if $M(n) = N$ and $M(s) = S$, we can systematically generate the vector space for transitive verbs of type $n^r sn^l$ as being $M(n^r sn^l) = M(n^r) \otimes M(s) \otimes M(n^l) = N \otimes S \otimes N$.

Next, we consider the full subcategory $\mathbf{M}_{P \times \mathbf{FVect}}$ of $P \times \mathbf{FVect}$. A subcategory of a category \mathbf{C} is a category \mathbf{C}' where the objects of \mathbf{C}' are objects of \mathbf{C} (hence $ob(\mathbf{C}') \subseteq ob(\mathbf{C})$), and likewise the morphisms of \mathbf{C}' are also morphisms of \mathbf{C} (hence $hom(\mathbf{C}') \subseteq hom(\mathbf{C})$). A subcategory is full if for all X, Y in $ob(\mathbf{C}')$, the sets $\{f \mid f : X \rightarrow Y \wedge f \in hom(\mathbf{C}')\}$ and $\{f \mid f : X \rightarrow Y \wedge f \in hom(\mathbf{C})\}$ are equal, i.e. if all morphisms between X and Y in \mathbf{C} are

also morphisms in \mathbf{C}' . The subcategory $\mathbf{M}_{P \times \mathbf{FVect}}$ of $P \times \mathbf{FVect}$ is defined as follows:

- For all $(a, A) \in ob(P \times \mathbf{FVect})$, $(a, A) \in \mathbf{M}_{P \times \mathbf{FVect}}$ if and only if $M(a) = A$.
- For all morphisms $(f, g) : (x, X) \rightarrow (y, Y)$ in $hom(P \times \mathbf{FVect})$, $(f, g) \in hom(\mathbf{M}_{P \times \mathbf{FVect}})$ if and only if (x, X) and (y, Y) are in $ob(\mathbf{M}_{P \times \mathbf{FVect}})$, and for $f = f_1 \otimes \dots \otimes f_n$ and $g = g_1 \otimes \dots \otimes g_m$ the following conditions hold: $m = n$, and g_i is a structural morphism of \mathbf{FVect} if f_i is a structural morphism of P .

The first condition ensures that we only consider the object pairings relevant to our task of associating syntax with semantics, and the second condition states that the subcategory is full, and places restrictions on which morphisms exist in the subcategory. We can verify that this category is still compact closed by the same reasoning used in §3.3.2, namely that if $f : x \rightarrow y$ is a structural morphism of P and $g : X \rightarrow Y$ is a structural morphism of \mathbf{FVect} , then if (x, X) and (y, Y) are in $ob(\mathbf{M}_{P \times \mathbf{FVect}})$, $(f, g) : (x, X) \rightarrow (y, Y)$ is also a structural morphism of $\mathbf{M}_{P \times \mathbf{FVect}}$, by virtue of the subcategory being full.

The map M ensures that for every $a \in ob(P)$ there is a unique $(a, A) \in ob(\mathbf{M}_{P \times \mathbf{FVect}})$, and that for every morphism $f \in hom(P)$ there is a unique morphism $(f, g) \in hom(\mathbf{M}_{P \times \mathbf{FVect}})$ by virtue of the fact that all morphisms in P are composed of structural morphisms, and that there is an injective map from the set of structural morphisms in P to those of \mathbf{FVect} . It follows that there is a functor $F : P \rightarrow \mathbf{M}_{P \times \mathbf{FVect}}$ defined as above, using M .

Furthermore, there is a trivial forgetful functor $G : \mathbf{M}_{P \times \mathbf{FVect}} \rightarrow \mathbf{FVect}$ which associates any object $(a, A) \in ob(\mathbf{M}_{P \times \mathbf{FVect}})$ to $A \in \mathbf{FVect}$, and any morphism $(f, g) \in hom(\mathbf{M}_{P \times \mathbf{FVect}})$ to a morphism $g \in \mathbf{FVect}$. It follows that the functor $G \circ F$ maps the objects and morphisms of P to those of \mathbf{FVect} . I have therefore shown that from the product category $P \times \mathbf{FVect}$ and a mapping M between pregroup types and vector spaces, a functor between the pregroup category P and the category of vector spaces \mathbf{FVect} can naturally be defined.

4.2 Supporting Context Free Grammars

While the term “context free grammar” technically refers to any grammar which is context free in the language theoretic sense, computational linguists usually refer to a specific sort of phrase-structure-grammar when talking about “Context Free Grammars” (CFGs). In this sense, the expression refers to a well-known and well-studied syntactic formalism for analysing the phrase structure of sentences, formalised by Chomsky in [11]. As a topic, it is covered in virtually any introduction to language theory, computational linguistics, or mathematical linguistics. Efficient parsing algorithms have been developed to recover the

CFG structure from digital text (e.g. CYK [15, 50, 89], Earley [25]), and a large number of corpora and other resources have been created to assist the learning of such generative grammars (e.g. the Penn Treebank [60]). It is a popular tool for computational linguistics, and it would make little sense to develop a formalism for syntactically motivated compositional distributional semantics that does not offer the opportunity to leverage the vast amounts of work done on and with CFGs.

In this section, I introduce a way of integrating context free grammars into the DisCoCat formalism described in Chapter 3. I begin, in §4.2.1 by briefly introducing CFGs and specifically the sort of CFGs I will be discussing later in this section. In §4.2.3, I then show how a CFG can be given a simple categorical structure, and how a functor from this category to \mathbf{FVect} can be defined by showing that there exists at least one functor between every CFG-as-a-category and some pregroup category P , guided by the fact that pregroup grammars and CFGs are weakly equivalent [9].

4.2.1 Context Free Grammar

In this section, I begin by giving a general definition of context free grammar, followed by the specification of some restrictions on CFGs which will be useful when giving CFGs categorical structure.

4.2.1.1 General Definition

Formally, a context free grammar is defined by four elements:

- a finite set V of non-terminal symbols, which can be seen as being the syntactic types of words and phrases. In simple grammars, these are usually given intuitive names such as NP for noun-phrases, VP for verb phrases, S for sentences, and so on.
- a set Σ of terminals, which we can interpret, within the context of linguistics, as being the words of our language.
- a finite set R of production rules, usually written in the form $A \Rightarrow B C$, $A \Rightarrow B$, $A \Rightarrow B C D$ and so on, where B , C and D may be elements of V or Σ , but A must be an element of V since it ‘generates’ the other symbols (and therefore is a non-terminal).
- the starting symbol of the grammar (from the set V). We will use S as starting symbol in this section, but there is no obligation to do so.

Rules of the form $A \Rightarrow B C$ can be read as “A generates the types B C”, or, going the other way, as saying that “a B and a C combine to form an A”. On the one hand, the generative direction from “top” to “bottom” is usually followed when parsing a string to assign to each word a syntactic type. The “bottom-to-top” approach, on the other hand, is more akin to what we do with pregroups when the types of words are known, and the CFG can be used to describe the phrasal structure of a sentence. Going this way, the generative rules can be seen as substitution rules where the element on the left of the arrow is substituted for the elements on the right to determine the type of the phrase comprising them.

S	\Rightarrow	$NP VP$
VP	\Rightarrow	$V_t NP$
NP	\Rightarrow	$Det N$
Det	\Rightarrow	the
N	\Rightarrow	dog cat
V_t	\Rightarrow	chases

Table 4.1: A simple CFG.

To give an example, a simple CFG is shown in Table 4.1. The set of non-terminal symbols is $V = \{S, NP, VP, V_t, Det\}$, the set of terminal symbols is $\{\text{the, dog, cat, chases}\}$, and the production rules are as shown. The $N \Rightarrow \text{dog|cat}$ is used as shorthand for the definition of several rules (namely $N \Rightarrow \text{dog}$ and $N \Rightarrow \text{cat}$) on one line. A parse tree for the sentence “The dog chases the cat.” is shown in Figure 4.1. Such parse trees are generated from the grammar’s production rules by interpreting each rule as the formation of a tree with, as root, the symbol on the left of the arrow; and with, as children of the root, the symbols on the right of the arrow, themselves being the roots of subtrees of one or more nodes. In this case, we see that ‘the’ and ‘dog’, as well as ‘the’ and ‘cat’, both being determiner-noun pairs $DT N$, each combine to form a small tree with a noun phrase NP as a root. The second of these noun phrases (“the cat”) combines with the transitive verb V_t , namely “chases”, to form a tree with a verb phrase VP as a root. This VP combines with the first NP to produce a tree with a sentence S as root, which is the whole parse tree for this sentence.

As a notational convenience, we state that if for some symbols A, B_1, \dots, B_n there exists a set of rules such that A generates $B_1 \dots B_n$, we write $A \Rightarrow^+ B_1 \dots B_n$ and call this a substitution sequence. We treat such substitution sequences as being uniquely characterised by a set of production rules used to generate $B_1 \dots B_n$ from A . In this sense, each distinct $A \Rightarrow^+ B_1 \dots B_n$ stands for a different subtree with A as root and B_1, \dots, B_n

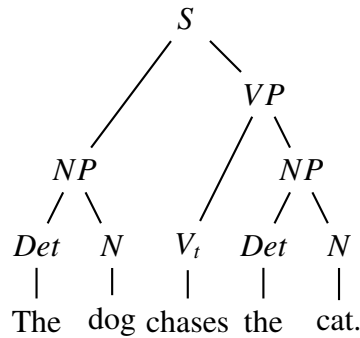


Figure 4.1: Sample parse tree for the CFG in Table 4.1.

as leaves. Naturally, any two substitution sequences $A \Rightarrow^+ B$ and $B \Rightarrow^+ C$ can be combined to form a unique substitution sequence $A \Rightarrow^+ C$.

To give a short example, in the sample CFG in Table 4.1, we can see that there exists a substitution sequence $S \Rightarrow^+ NP V_t NP$ because $S \Rightarrow NP VP$ and $VP \Rightarrow V_t NP$.

4.2.1.2 Restrictions

While the above description of CFGs suffices to fully define the basic mechanics and components of phrase-structure CFGs, I add a further four restrictions on the form of the CFGs we will consider in this section. The reason for these restrictions will be explained at the end of §4.2.3.2.

First, we will only consider pseudo-proper CFGs. A CFG is pseudo-proper if:

- Every non-terminal except I is on the left-hand side of some production rule;
- For every terminal or non-terminal, there is some sequence of production rules such that the root symbol S generates a set of symbols including that terminal or non-terminal;
- There are no cycles, i.e. for any symbol A there is no sequence of production rules $A \Rightarrow^+ A$ beginning with that symbol on the left and ending with that symbol as the sole symbol on the right.
- Any sequence of symbols containing the empty string ε is equivalent to the same sequence of symbols with all instances of ε removed. Any sequence of symbols is trivially equivalent to itself.

These rules entail that for any sequence of symbols, there is at most a finite set of finite trees with S as root that form valid parses of that sequence. I call these CFGs pseudo-

proper because of the last rule about ε . The definition of proper CFGs replaces this rule with one stating that the empty string will not be on the right-hand side of any production rule, as a way of ensuring that there are a finite number of trees with a finite sequence of symbols as leaves and S as a root. The modification of this rule which I introduced seeks to preserve this property without removing ε from our list of symbols, as it will be useful in the next section. The last rule instead simply equates all sequences of symbols which are identical once ε is removed, thereby treating it as an ‘invisible’ symbol potentially present in any sequence, and effectively eliminating it from the parsing process.

Second, we only consider CFGs that produce binarised parsed trees, meaning that every production rule has at least one symbol on the right of the arrow, and at most two. We do not count the ‘invisible’ symbol ε in this restriction. This restriction can be applied without loss of generality, as any CFG can be modified to fit this constraint by introducing a finite number of additional symbols while still recognising the same language. For any rules of the form

$$A \Rightarrow B_1 B_2 \dots B_n$$

I introduce a new symbol C_n and a rule

$$C_n \Rightarrow B_1 B_2 \dots B_{n-1}$$

and produce an amended version of the first rule

$$A \Rightarrow B_1 C_n$$

We apply this rewrite rule recursively to the newly produced rule in order to produce symbols C_{n-1} , C_{n-2} , etc. and associated production rules, until we obtain some symbol C_i which has as production rule

$$C_i \Rightarrow B_1 B_2$$

at which point the original rule has been fully binarised. To give a short example, assume we have the rule

$$A \Rightarrow B C D E$$

in our CFG. We begin by rewriting it as

$$A \Rightarrow F E$$

where F is a newly introduced symbol. We then introduce the rule

$$F \Rightarrow B C D$$

which is not binarised, so we must rewrite it as

$$F \Rightarrow G D$$

where G is newly introduced symbol. We then introduce the rule

$$G \Rightarrow B C$$

which is binarised, so the rewrite process is complete. We have therefore binarised the rule

$$A \Rightarrow B C D E$$

by replacing it with the set of rules

$$A \Rightarrow F E$$

$$F \Rightarrow G D$$

$$G \Rightarrow B C$$

This rewrite system is part of the process of translating any grammar to Chomsky Normal Form (CNF), although the thus-produced binarised CFG is not technically in CNF, as we allow rules for the form

$$A \Rightarrow B$$

where B is a non-terminal (whereas a CFG in CNF has only binary production rules with non-terminals on the right, and unary production rules with terminals on the right). The point of binarising a grammar will be made clear in §4.2.3 when we discuss the categorical structure of CFGs, but full-blown translation into CNF is not required.

Third, we will disallow production rules which have a mix of terminals and non-terminals, excluding ε , on the right-hand side of the arrow. This assumption can be made without loss of generality as any rule with such a mixture of non-terminals and terminals can be rewritten by replacing each terminal symbol with a new non-terminal, and introducing a new production rule in which the newly created non-terminal produces the terminal symbol.

Finally, the fourth restriction is that for any non-terminal which appears on the left-

hand side of a production rule with non-terminals on the right-hand side, there is no rule with that symbol on the left-hand side and non-terminals, excluding ε , on the right-hand side. Again, this restriction can be applied without loss of generality: we modify any rule violating the restriction, and of the form

$$A \Rightarrow n$$

where n is one or more non-terminals (in the form $n_1|n_2|\dots|n_k$), by rewriting it as

$$A \Rightarrow N$$

where N is a new non-terminal symbol, and adding the rule

$$N \Rightarrow n$$

We call such non-terminals, which are on the left-hand side of production rules with terminals on the right-hand side, *basic types*. All other non-terminals will be referred to as *complex types*. Rules with basic types on the left-hand side will be referred to as *terminal rules*, while rules with complex types on the left-hand side will be referred to as *non-terminal rules*. We call R_{NT} the subset of R consisting of all the non-terminal rules.

4.2.2 CFGs as Categories

To use CFGs in the DisCoCat formalism instead of pregroup grammars, two things must be done. First, we need to show that CFGs can be given a categorical interpretation. Second, we need to show that there is a functor between any CFG-as-a-category and the category **FVect**.

Here, I show that any CFG satisfying the restrictions described above can be interpreted as a category **CFG**:

1. Let V be the set of non-terminals of our CFG.
2. Let $ob(\mathbf{CFG})$ be a set such that $V \subset ob(\mathbf{CFG})$ and $\varepsilon \in ob(\mathbf{CFG})$.
3. Let the sequence of symbols of form $A B$ found on the right-hand side of production rules in a CFG be the concatenation of the objects A and B .
4. Let the concatenation operation for types be the associative monoidal tensor \otimes in the category, such that for any A and B in $ob(\mathbf{CFG})$, the object $A \otimes B$ is in $ob(\mathbf{CFG})$.

5. Let $hom(\mathbf{CFG})$ be a set of arrows, where for any substitution sequence $A \Rightarrow^+ B$ there is one unique arrow $f : B \rightarrow A$ in $hom(\mathbf{CFG})$.
6. Let $A \otimes \varepsilon \cong A \cong \varepsilon \otimes A$ for any $A \in ob(\mathbf{CFG})$, by the rule for equality of sequences containing ε , defined above.
7. Let self equivalence of sequences of symbols define an arrow from each object to itself.

I hypothesise that these stipulations define a monoidal category \mathbf{CFG} . Stipulations 1–4 define the set of objects in the category. Stipulation 5 defines the set of morphisms in the category. Stipulations 4 and 6 state that if \mathbf{CFG} is a category, then it is a monoidal category with ε as unit. To confirm that this is a valid definition of a category, we verify that it satisfies the axioms outlined in §3.2.1:

- By the definition of substitution sequences, any two sequences $A \Rightarrow^+ B$ and $B \Rightarrow^+ C$ can be combined to form a substitution sequence $A \Rightarrow^+ C$. This combination corresponds to arrow composition: for $A \Rightarrow^+ B$ there is some arrow $f : B \rightarrow A$; for $B \Rightarrow^+ C$ there is some arrow $g : C \rightarrow B$; for $A \Rightarrow^+ C$ there is some arrow $h : C \rightarrow A$, and the combination of substitution sequences states $f \circ g = h$.
- Substitution sequences are uniquely characterised by the production rules involved, hence if $A \Rightarrow^+ B$ and $B \Rightarrow^+ C$ combine to form a sequence $A \Rightarrow^+ C$, and $B \Rightarrow^+ C$ and $C \Rightarrow^+ D$ combine to form a sequence $B \Rightarrow^+ D$, then the combinations of $A \Rightarrow^+ B$ with $B \Rightarrow^+ D$ and $A \Rightarrow^+ C$ with $C \Rightarrow^+ D$ form the same sequence $A \Rightarrow^+ D$. With morphisms $f : B \rightarrow A$ as $A \Rightarrow^+ B$, $g : C \rightarrow B$ as $B \Rightarrow^+ C$ and $h : D \rightarrow C$ as $C \Rightarrow^+ D$, then $(f \circ g) \circ h = f \circ (g \circ h)$, and therefore arrow composition is associative.
- The self equivalence arrows $f_A : A \rightarrow A$ from any object A to itself correspond to replacing a sequence of symbols with itself. Doing this before applying a substitution sequence $A \Rightarrow^+ B$ associated with some morphism $g : B \rightarrow A$ does not change the outcome of the substitution, hence $g \circ f_A = g$. Furthermore, because the replacing of the sequence of symbols B with itself after the application of a substitution sequence $A \Rightarrow^+ B$ does not change the outcome of the substitution sequence, it follows that $f_B \circ g = g$. It follows that self equivalence arrows satisfy the identity arrow axiom.

So we have a collection of objects, a collection of morphisms, and a composition operation, all of which satisfy the three axioms defining a category. Furthermore, we have an identity object and a monoidal tensor, so it therefore follows that any \mathbf{CFG} defined from a context free grammar satisfying the restrictions outlined in §4.2.1 is a monoidal category.

4.2.3 Defining a Functor

We now turn to the task of defining a functor between some CFG-as-a-category **CFG** and **FVect**. Contrary to what we did with pregroup categories P where we relied on the compact closed nature of both P and **FVect** to define a functor, things are not as simple with **CFG**. While with pregroup categories, the structure of pregroup types dictates that a transitive verb of form $n^r sn^l$ be mapped to a vector space $N \otimes S \otimes N$, nothing about the transitive verb ‘type’ in CFGs tells us that we should map V_i to something of the structure $N \otimes S \otimes N$ rather than any other object in **FVect**. Likewise for arrows: the structure of pregroup reduction morphisms in P dictates the structure of the morphisms they are mapped to in **FVect**, such that for example $\epsilon_n^r \otimes 1_s \otimes \epsilon_n^l : n^r sn^l \rightarrow s$ in $\text{hom}(P)$ should map to $\epsilon_N \otimes 1_S \otimes \epsilon_N : N \otimes S \otimes N \rightarrow S$ in $\text{hom}(\mathbf{FVect})$ rather than some other linear map between $N \otimes S \otimes N$ and S . In short, contrary to pregroups which directly reveal their ‘functional structure’ for both objects and reductions, CFGs conceal it, and an additional step must be taken before defining a functor between some category **CFG** and **FVect**. Here, I will show how for every **CFG** there exists one or more pregroup categories P_i such that a functor F_i can be systematically defined between **CFG** and each P_i , thereby allowing functorial passage from **CFG** to **FVect** via some P_i . I do this by first presenting an algorithm translating any CFG with $|R_{NT}|$ non-terminal production rules into at most $2^{|R_{NT}|}$ pregroup grammars. Second, I will show that each such CFG-to-pregroup grammar translation constitutes a functorial passage between a CFG-as-a-category **CFG** and some pregroup category P .

4.2.3.1 CFG to Pregroup Translation

Inner Structure To translate a CFG to a pregroup, we must “reveal”, or rather make assumptions about, the inner structure of types and generative rules. In other words, we wish to treat a production rule as a type reduction, and spell out the mechanisms of such a reduction, in order to add compact closure to the categorical representation of the grammar. Recalling that for each production rule in a CFG of the form

$$A \Rightarrow B C$$

there is some arrow f in the categorification of that CFG such that $f : B \otimes C \rightarrow A$, the simplest way to reinterpret this in a compact closed setting is to assume that one of the types B or C contains the adjoint of the other, and the type of A . Stating this explicitly, let’s assume that CFG type A corresponds to some pregroup type a . Then either C corresponds to some pregroup type c and B to some compound type ac^l , in which case $f : B \otimes C \rightarrow A$ in $\text{hom}(\mathbf{CFG})$ corresponds to $1_a \otimes \epsilon_c^l : (ac^l)c \rightarrow a$ in $\text{hom}(P)$; or B corresponds to some

pregroup type b and C to some compound type $b^r a$, in which case $f : B \otimes C \rightarrow A$ in $\text{hom}(\mathbf{CFG})$ corresponds to $\epsilon_b^r \otimes 1_a : b(b^r a) \rightarrow a$ in $\text{hom}(P)$.

It is important to note that these two ways of translating a production rule make no assumptions about whether the types which are *not* inferred to be compound types—namely a and c in the first case, and a and b in the second—are compound or simple types themselves. For example, if we have already inferred that A must be of some pregroup type $a = d^r e$, then the type of B in the first case will be $ac^l = d^r ec^l$, and the type of C in the second case will be $b^r a = b^r d^r e$. Similarly, if we had previously inferred the pregroup type of C to be $c = po^l$, then the type of B in the first case becomes $ac^l = a(po^l)^l$, and similarly for the type of C in the second case if that of B had previously been inferred.

Simple Type Inference Since each production rule in a binarised parse tree has at most two elements on the right-hand side, and each such rule generates two options in terms of pregroup interpretation of the CFG types, we immediately get the result that each CFG with $|R_{NT}|$ production rules generates at most $2^{|R_{NT}|}$ different pregroup translations using this process. At first blush, it seems that all we need to do is set some convention such as always interpreting the left element of the pair of types on the right-hand side of each production rule as being a compound type in order to obtain a canonical pregroup translation of a CFG. For example, we could always treat a generative rule of the form

$$A \Rightarrow B C$$

as a pregroup reduction of the form

$$(ac^l)c \rightarrow a$$

hence interpreting A as some type a , B as some type ac^l and C as some type C . However, translating a CFG into a pregroup is not quite so straightforward, as there are two cases where a rule of the form

$$A \Rightarrow B C$$

may only be translated in one way.

The first case, where $A = B$ or $A = C$, is fairly trivial. If $A = B$, then we cannot interpret B as ac^l , since this would entail $a = ac^l$, which generates an infinite type if $c \neq 1$, as $a = ac^l = ac^l c^l = ac^l c^l c^l$ and so on. Hence if $A = B$, then the rule must be interpreted as $a(a^r a) \rightarrow a$, and so B must be translated as a , and C must therefore be translated as $a^r a$. By a similar argument, if $A = C$, then the rule must be interpreted as $(aa^l)a \rightarrow a$, C must

be translated as a , and B must therefore be translated as aa^r .

The second case is a little more subtle, and regards the fairly frequent situation where a non-terminal symbol appears on the right-hand side of more than one rule. Let us assume that the following two rules are in our CFG:

$$\begin{aligned} A &\Rightarrow B C \\ D &\Rightarrow B E \end{aligned}$$

If we infer from the first rule that the pregroup types of A and C are a and c respectively, and that B has the compound type ac^l , then we cannot make a new type inference for B in the second case whereby B is a compound type. If we could, then we would, assuming D and E have pregroup types d and e respectively, obtain the following equality:

$$ac^l = de^l$$

which holds only for the case where $a = d$ and $c = e$. A similar argument holds for B in the pairings:

$$\begin{aligned} A &\Rightarrow B C \quad \text{and} \quad D \Rightarrow E B \\ A &\Rightarrow C B \quad \text{and} \quad D \Rightarrow B E \\ A &\Rightarrow C B \quad \text{and} \quad D \Rightarrow E B \end{aligned}$$

thereby covering all cases without loss of generality.

Type Inference Algorithm Considering both of these cases and formalising them, we can produce an algorithm, shown in Figure 4.2, which takes a CFG and outputs a set of dictionaries each mapping the symbols of the CFG to pregroup types in a pregroup grammar P . These dictionaries then allow for the production rules of the CFG to be interpreted as reduction rules in P .

Let's unpack the algorithm shown in Figure 4.2 a little. Step 1 defines an initial injective mapping from CFG symbols to hypothetical types in a pregroup, assuming that each CFG non-terminal stands for a unique pregroup type. This mapping, represented as a type dictionary, is used to initialise the algorithm, but also during the running of the algorithm to verify whether or not a type has been already been set by a rule (i.e. whether or not the pregroup type assigned to a CFG non-terminal has been changed since the beginning of the algorithm). We also assign the empty string ε to the pregroup unit 1, an assignment which will not be changed during the algorithm.

1. Let D_0 be a dictionary mapping each non-terminal X in a CFG to a unique symbol $D_0[X] = x$. Let $D_0[\varepsilon] = 1$.
2. Let B be a list initially containing only D_0 .
3. Let R_{NT} be a list of CFG production rules named R_1, R_2, \dots
4. If R_{NT} is not empty, remove the first R_i element for R , otherwise terminate and return B .
5. Let B' be an empty list. For each $D_j \in B$ follow these steps:
 - (a) Let D_k, D_m be new dictionaries.
 - (b) If R_i is of the form $A \Rightarrow B$, then let $D_k[X] = D_j[X]\langle D_j[B] := D_j[A] \rangle$ for all X in D_j , where $v\langle w := y \rangle$ means that all instances of w in v are to be replaced with y , and add D_k to B' .
 - (c) If R_i is of the form $A \Rightarrow B C$ and $D_k[B] = D_0[B]$ and $B \neq A$, then let $D_k[X] = D_j[X]\langle D_j[B] := D_j[A] \cdot D_j[C] \rangle$ for all X in D_j and add D_k to B' .
 - (d) If R_i is of the form $A \Rightarrow B C$ and $D_k[C] = D_0[C]$ and $C \neq A$, then let $D_m[X] = D_j[X]\langle D_j[C] := D_j[B]^r \cdot D_j[A] \rangle$ for all X in D_j and add D_m to B' .
6. Let $B = B'$. Go to step 4.

Figure 4.2: Procedure for translating a CFG with $|R_{NT}|$ production rules into $k \leq 2^{|R_{NT}|}$ pregroup grammars.

Step 2 defines a boundary list. As the algorithm is effectively constructing a tree, this list can be seen as containing all the leaves in the tree at any particular point in the algorithm's running. Step 3 defines a list of production rules R_{NT} which we will use as a stack, and which initially contains all the non-terminal rules in the CFG. Step 4 forms the beginning of a loop which runs while R_{NT} is not empty, and which at each iteration pops a new production rule off the R_{NT} stack and runs the steps which follow. If the stack is empty, it returns the boundary list B which contains the leaves of the fully constructed tree.

Steps 5 is the body of a loop run for each new rule R_i popped off the R_{NT} stack, iterating over the CFG-to-pregroup type mapping dictionaries in the boundary B . It begins by creating an empty list B' which will become the new boundary (as we are turning leaves in the old boundary into branch nodes). Step 5a creates two new empty dictionaries D_k and D_m for each dictionary D_j in the old boundary. If the R_i is a unary rule (one non-terminal), step 5b is run. Step 5b interprets rules of the form $A \Rightarrow B$ as mappings $b \rightarrow a$ where b and a are obtained from $D_j[B]$ and $D_j[A]$. It copies all the mappings in D_j to the new dictionary D_k while replacing every instance of b in the value of D_k with a . This new dictionary is

added to the new boundary B' , discarding D_m . If the rule is binary, steps 5c and 5d are run. Step 5c interprets rules of the form $A \Rightarrow B C$ as mappings $(ac^l)a \rightarrow a$ where c and a are obtained from $D_j[C]$ and $D_j[A]$. If non-terminals B and A do not use the same symbol (i.e. $B \neq A$), and if the pregroup type of B has not already been defined by some other rules (i.e. $D_k[B] = D_0[B]$, namely if the pregroup interpretation of B is still the same as the initial assignment in D_0), then the mappings of D_j are copied to the new dictionary D_k , replacing all instances of $b = D_0[B]$ with ac^l , and then adding D_k to the new boundary list B' . Similarly, step 5c interprets rules of the form $A \Rightarrow B C$ as mappings $bb^r a \rightarrow a$ where b and a are obtained from $D_j[B]$ and $D_j[A]$. If non-terminals C and A do not use the same symbol (i.e. $C \neq A$), and if the pregroup type of C has not already been defined by some other rules (i.e. $D_k[C] = D_0[C]$, namely if the pregroup interpretation of C is still the same as the initial assignment in D_0), then the mappings of D_j are copied to the new dictionary D_m , replacing all instances of $c = D_0[C]$ with $b^r a$, and then adding D_m to the new boundary list B' .

Finally, step 6 sets the boundary B to the newly produced boundary list B' by redefining B as being equal to B' . The algorithm then returns to step 4.

Revisiting this algorithm in even more general terms, steps 1–3 initialise a tree construction algorithm. Step 4 creates a loop over non-terminal rules in the CFG, creating a new set of nodes to be added to leaves of the tree. And step 5 does the actual work of creating new nodes for each leaf of the tree, the number of which depends on the structure of the CFG rule being considered, and whether or not the types being inferred have already been inferred by a previous rule, or whether there are restrictions on what inferences can take place. Step 6 completes the loop started in step 4.

Algorithm Output The thus constructed tree starts with a single type dictionary assuming every CFG terminal corresponds to a unique atomic pregroup type. As we progress down the tree, nodes correspond modified copies of their parents where the pregroup structure of CFG types is progressively inferred by considering the pregroup interpretations of CFG rules. Each branch node has one or more children generated by different interpretations of the CFG rule being considered at that depth. Left children correspond to the inference made from the interpretation of unary rules and of binary rules where the left element of the right-hand side is interpreted as a compound type; while right children correspond to the inference made from the interpretation of binary rules where the right element of the right-hand side is interpreted as a compound type. The leaves of the tree are the type dictionaries inferred once all the CFG non-terminal rules have been considered.

Algorithm Complexity Let $m = |R_{NT}|$ be the number of production rules in the grammar being processed by this algorithm, and $n = |V|$ be the number of non-terminals. The algorithm creates a binary tree of depth m where each node costs n operations to construct. The time complexity of the algorithm is therefore $O(m^2n)$.

The algorithm can be modified to run in $O(mn)$ time by skipping to step 6 once one of steps 5b, 5c, or 5d has been executed (i.e. produced a new leaf). This corresponds to a dynamic algorithm descending the tree produced by the full-blown algorithm by following the left-most path.

Algorithm Termination As a byproduct of the complexity analysis, it is easy to see that the algorithm is guaranteed to terminate: each loop in Step 5 changes the definition of exactly one of the remaining unchanged types, and hence the loop will run at most n times before terminating the algorithm. The algorithm is furthermore guaranteed to succeed if the CFG satisfies the restrictions presented above, as is discussed at the end of this section.

4.2.3.2 From Translation Dictionaries to Functors

We now complete this section by showing that each of these dictionaries $D_i \in B$ is sufficient to define a functorial passage $F_i : \mathbf{CFG} \rightarrow P_i$ from the categorification \mathbf{CFG} of a CFG to some pregroup P_i associated with D_i . Each dictionary D_i assigns to each CFG non-terminal symbol a pregroup type in some pregroup P_i , freely generated by the set of atomic symbols in the dictionary $\{D_i[X] | X \in D_i \wedge D_i[X] = D_0[X]\}$. Therefore we obtain our functorial map for objects:

$$F_i : ob(\mathbf{CFG}) \rightarrow ob(P_i) :: X \mapsto D_i[X]$$

which covers symbol concatenation as follows:

$$F_i : ob(\mathbf{CFG}) \rightarrow ob(P_i) :: X \otimes_{\mathbf{CFG}} Y \mapsto D_i[X] \otimes_{P_i} D_i[Y]$$

Therefore if some D_i defines

$$D_i[V_i] = \bar{n}^r s \bar{n}^l$$

$$D_i[N] = n$$

$$D_i[Det] = \bar{n} n^l$$

then the CFG parse in of a sentence such as “The dog chased the cat”, namely $S \Rightarrow^+ Det N V_i Det N$, would translate via F_i to $(\bar{n} n^l) n (\bar{n}^r s \bar{n}^l) (\bar{n} n^l) n$ in P_i , which reduces to s .

This brings us to the second aspect of the functor, which is the map between morphisms which would allow us to formalise how a parse in a CFG maps to a reduction operation in P_i . We essentially get this “for free” by the uniqueness up to isomorphism of morphisms between any two objects in any pregroup category P_i , since the morphisms stand for ordering relations. The idea is then that for any morphism $f : A \rightarrow B$ in $\text{hom}(\text{CFG})$ we map f to the unique morphism $g = F(f) : F(A) \rightarrow F(B)$ in P_i . This presupposes that there is a morphism $g : F(A) \rightarrow F(B)$ in $\text{hom}(P_i)$ when there is some $f : A \rightarrow B$ in $\text{hom}(\text{CFG})$, but this is precisely what the algorithm I presented here ensures: we assign pregroup types to CFG non-terminals by exploiting the fact that whatever the concatenation of symbols on the right of the production rule is translated as reduces to whatever the translation of the symbol on the left is. This is to say that for every concatenation of CFG symbols in the grammar there is some morphism in P_i which stands for the reduction of their corresponding pregroup types.

Throughout this section we have been talking about non-terminals. However, an essential part of a pregroup grammar is the type dictionary for terminal symbols which assigns to each word in our language one or more pregroup types. We get this type dictionary for free from the pregroup translation dictionaries defined above and the terminal rules in our CFG. We simply state that for every map D_i translating the non-terminals and productions of some context free grammar into the types and reductions of some pregroup grammar P_i there exists a term-to-type dictionary T_i . This term-to-type dictionary is defined as follows: for any terminal rule in the CFG of the form

$$A \Rightarrow w_1|w_2|\dots|w_n$$

producing one or more terminal symbols from a non-terminal A , the pregroup type of those non-terminals is simply the type of the non-terminal producing them; in other words:

$$T_i[w_1] = D_i[A] \quad T_i[w_2] = D_i[A] \quad \dots \quad T_i[w_n] = D_i[A]$$

To conclude this section, let’s examine why the restrictions on CFGs presented in §4.2.1.2 were helpful in defining this algorithm and functorial passage:

1. We only considered pseudo-proper CFGs so that:
 - All non-terminals (except I) are on the left-hand side of some rule, allowing us to infer the type of all non-terminals with complex types, and produce the type dictionaries for the basic types.
 - All symbols are connected to the root S by some sequence of rules, ensuring

once again that each type can be inferred.

- The prohibition of cycles ensures that the algorithm does not reach a deadlock.
 - Equivalence under concatenation with ε allows us to use ε as special ‘empty’ type, purely so it can be mapped to the unit object in a pregroup category.
2. Only considering binarised rules allowed for a fairly simple algorithm. This isn’t necessary *per se*, but was set as a restriction principally out of convenience.
 3. Disallowing production rules with a mix of terminals and non-terminals on the right-hand side served to separate type inference from the creation of a type dictionary mapping words to sets of basic types.
 4. The restriction prohibiting a symbol from being on the left-hand side of both rules with non-terminals on the right and rules with terminals on the right served to create a distinction between basic and complex types, which was, once again, done to separate type inference from the production of the type dictionary.

4.3 Supporting Lambek Grammar

Another grammatical formalism which might be useful to support in the DisCoCat framework is the categorial grammar developed by Lambek in the 50s, generally referred to as Lambek Grammar. Aside from showcasing the ability of DisCoCat to adapt to a wide-range of grammatical systems, supporting Lambek Grammar has the additional advantage of laying groundwork for supporting a more expressive categorial grammar, namely Combinatorial Categorial Grammar (CCG), which I will discuss in §7.3. In this section, I will briefly present Lambek Grammar, and the sort of category it will be represented as, before discussing how a functorial passage can be defined from this category to the category of vector spaces \mathbf{FVect} .

4.3.1 Lambek Grammar

Lambek Grammar (LG), first described in [51], is a context free categorial grammar consisting of an infinite set of types, and a set of type operations, which allow for the analysis of syntax to be completed in a manner akin to logical proofs.

Types in LG are recursively defined as follows:

- Atomic types are represented as single letters A, B, \dots

- $A \setminus_L B$ is a type if A and B are types¹.
- A/B is a type if A and B are types.
- $A \cdot B$ is a type if A and B are types, with \cdot being associative.

(Axiom) $\frac{}{A \vdash A}$	(Cut) $\frac{\Gamma, A, \Pi \vdash B \quad \Delta \vdash A}{\Gamma, \Delta, \Pi \vdash B}$
(L \cdot) $\frac{\Gamma, A, B, \Pi \vdash C}{\Gamma, A \cdot B, \Pi \vdash C}$	(R \cdot) $\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \cdot B}$
(L \setminus_L) $\frac{\Gamma, B, \Pi \vdash C \quad \Delta \vdash A}{\Gamma, \Delta, A \setminus_L B, \Pi \vdash C}$	(R \setminus_L) $\frac{A, \Gamma \vdash B}{\Gamma \vdash A \setminus_L B}$
(L/ \setminus) $\frac{\Gamma, A, \Pi \vdash C \quad \Delta \vdash B}{\Gamma, A/B, \Delta, \Pi \vdash C}$	(R/ \setminus) $\frac{\Gamma, B \vdash A}{\Gamma \vdash A/B}$

Table 4.2: Axioms of Lambek Grammar.

The axioms of LG are shown in Table 4.2, in the form presented in [8]. These axioms allow ‘empty’ values for Γ , Δ and Π . We therefore assume the existence of an empty type I acting as a multiplicative unit for the operation \cdot such that $A \cdot I = A = I \cdot A$ for any A .

From these axioms, the following type operations are inferred [66, 29]:

- **Slash introduction:** If $A \cdot B \vdash C$ then $A \vdash C/B$ and $B \vdash A \setminus_L C$.
- **Slash elimination:** $(A/B) \cdot B \vdash A$ and $B \cdot (B \setminus_L A) \vdash A$.
- **Composition:** $A/B \cdot B/C \vdash A/C$ and $A \setminus_L B \cdot B \setminus_L C \vdash A \setminus_L C$.
- **Type-raising:** $A \vdash B/(A \setminus_L B)$ and $A \vdash (B/A) \setminus_L B$.

To complete the definition of a Lambek Grammar, a type dictionary is defined mapping each word in our lexicon to the set of types it can hold. A sentence $w_1 w_2 \dots w_n$ is grammatically correct if there is some T_i for each word w_i in the type dictionary such that $T_1 \cdot T_2 \cdot \dots \cdot T_n \vdash S$, where S the type associated with sentences, can be logically derived from the axioms of *LG*.

¹Here, I write backslashes with the subscript L to indicate that I am following Lambek’s slash notation, which differs from the slash notation commonly used in CCG, discussed in §7.3.

4.3.2 Lambek Grammars as Monoidal Bi-Closed Categories

Here, we show that a Lambek Grammar \mathbf{LG} can be modelled as a monoidal bi-closed category \mathbf{LG} , as was discussed and formalised by Bob Coecke, Mehrnoosh Sadrzadeh and myself in [17], and previously observed in the original work on Lambek Grammars by Lambek himself.

A monoidal bi-closed category is a monoidal category with unit I and associative monoidal tensor \otimes . What differentiates it from a normal monoidal category is that for every pair of objects A and B in the category, there are objects $A \multimap B$ and $A \multimap B$, and the morphisms

- $ev_{A,B}^l : A \otimes (A \multimap B) \rightarrow B$
- $ev_{A,B}^r : (A \multimap B) \otimes B \rightarrow A$
- $\Lambda^l(f) : C \rightarrow A \multimap B$ for any $f : A \otimes C \rightarrow B$
- $\Lambda^r(g) : C \rightarrow A \multimap B$ for any $g : C \otimes B \rightarrow A$

such that the following diagrams commute:

$$\begin{array}{ccc}
 A \otimes C & \xrightarrow{1_A \otimes \Lambda^l(f)} & A \otimes (A \multimap B) \\
 & \searrow f & \downarrow ev_{A,B}^l \\
 & & B
 \end{array}
 \qquad
 \begin{array}{ccc}
 C \otimes B & \xrightarrow{\Lambda^r(g) \otimes 1_B} & (A \multimap B) \otimes B \\
 & \searrow g & \downarrow ev_{A,B}^r \\
 & & A
 \end{array}$$

The morphisms ev^l and ev^r are referred to as evaluation morphisms, while the Λ morphisms Λ^l and Λ^r are called currying morphisms.

It is fairly straightforward to show that any LG can be represented as a closed monoidal category \mathbf{LG} which is free in its objects. The empty type I is the unit object I . For any atomic type A there is an object A in the category. For each type of the form $A \cdot B$ there is some object $A \otimes B$. For each type $A \backslash_L B$ there is an object $A \multimap B$ and for each type A / B there is an object $A \multimap B$. In short: the objects of the category are freely generated from the set of of atomic types. This feature will be essential to the definition of a functor between the categorical representation of Lambek Grammars and the category of vector spaces \mathbf{FVect} .

As for our type operations:

- **Slash introduction:** let $A \cdot B \vdash C$ be the morphism $f : A \otimes B \rightarrow C$. Slash introduction $B \vdash A \setminus_L C$ corresponds to the morphism $\Lambda^l(f) : B \rightarrow A \multimap C$, and $A \vdash C/B$ corresponds to the morphism $\Lambda^r(f) : A \rightarrow C \multimap B$.
- **Slash elimination** simply corresponds to the evaluation morphisms.
- **Composition:** for any pair of objects $A \multimap B$ and $B \multimap C$ in the category, there is a morphism

$$\text{comp}_{A \multimap B, B \multimap C}^r : (A \multimap B) \otimes (B \multimap C) \rightarrow A \multimap C$$

Likewise, for any pair of objects $A \multimap B$ and $B \multimap C$ in the category, there is a morphism

$$\text{comp}_{A \multimap B, B \multimap C}^l : (A \multimap B) \otimes (B \multimap C) \rightarrow A \multimap C$$

- **Type raising** morphisms $\text{raise}_{A,B}^l$ and $\text{raise}_{A,B}^r$ are just the right and left currying of the left and right evaluation morphisms:

- $\text{raise}_{A,B}^l = \Lambda^r(\text{ev}_{A,B}^l) : A \rightarrow B \multimap (A \multimap B)$
- $\text{raise}_{A,B}^r = \Lambda^l(\text{ev}_{B,A}^r) : A \rightarrow (B \multimap A) \multimap B$

We saw, in Chapter 3, that compact closed categories have a sound and complete graphical calculus allowing us to reason diagrammatically about structures within such categories. Do we need to give up on this powerful tool when dealing with closed monoidal categories as we do here? Fortunately not.

As I discussed in [17], with Mehrnoosh Sadrzadeh and Bob Coecke, a diagrammatic calculus for such categories has been developed by Baez and Stay [3]. Apart from depicting the flow of meaning, this calculus can also be applied to depict grammatical reduction of Lambek monoids, resembling the constituency parse trees obtained from context free grammars.

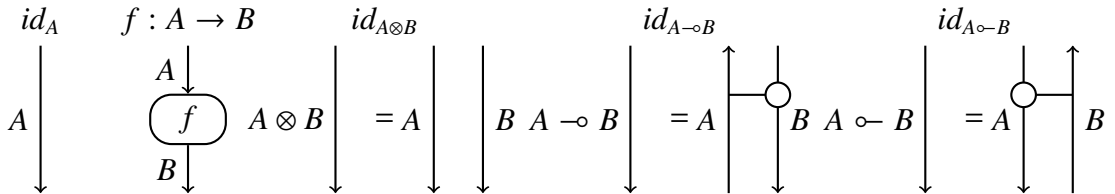


Figure 4.3: Basic diagrammatic language constructs.

The basic constructs of the diagrammatic language for monoidal bi-closed categories is shown in Figure 4.3. These are read from top to bottom, such that sequential composition of

morphisms in the category corresponds to the downwards extension of the diagram. Arrows are annotated with objects of the category, and morphisms are represented as ‘blobs’ with one or more ‘input’ arrows and one or more ‘output’ arrows standing for the domain and codomain. The tensoring of two objects corresponds to the side by side placement of their arrows. Topological equivalence between diagrams indicates an isomorphism between the corresponding categorical objects and vice versa, hence the diagram for $(A \otimes B) \otimes C$ is identical to that of $A \otimes (B \otimes C)$. Finally, there exists a rewrite rule for the objects of the form $A \multimap B$ and $A \multimap B$ as shown in the last two diagrams of Figure 4.3. The clasp is a restriction in the diagrammatic calculus which prevents us from treating both arrows as separate entities, e.g. such that a function cannot be applied to one and not to the other.

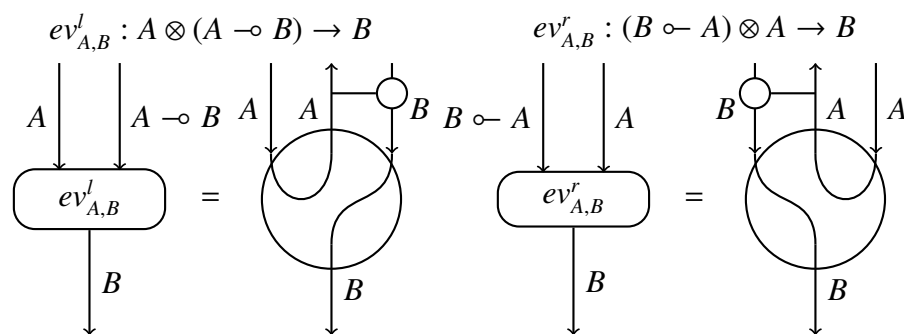


Figure 4.4: Rewrite rules for ev^l and ev^r .

Another more complex rewrite rule is that provided for the ev^l and ev^r morphisms, shown in Figure 4.4. These each take two arrows in and output one: the in arrow of the form $A \multimap B$ or $B \multimap A$ is rewritten in clasp form using the rewrite rules from Figure 4.3; and the ‘internal structure’ of the morphism is exposed by drawing a bubble around cups and twists that redirect the non-clasped input arrow into the upward arrow from the clasped pair with a cup. Figure 4.5 shows the diagrams for currying, taken from [3], which represent the slash introduction rules. Figure 4.6 shows the diagrams for the composition rules.

Finally, Figure 4.7 shows the diagrams for the type raising rules, obtained by currying the evaluation rules. This last figure may require a bit of explanation: note that the clasps on the bottom don’t seem to fully match the direction of the \multimap and \multimap operators in the mathematical definition shown above them. This is no error: since expressions of the form $A \multimap B$ and $B \multimap A$ are represented diagrammatically as an upwards pointing arrow labelled A clasped with a downward pointing arrow labelled B ; when A is of the form $C \multimap D$ or $D \multimap C$ we must in turn expand the upward pointing arrow into two clasped arrows. However, as the arrow we are ‘expanding’ is pointing upwards, we must turn the clasped arrow representations upside-down, thereby reversing the up/down direction of clasped

arrows and swapping their order (effectively rotating the clasped arrows by 180°). This means that if A is pointing upwards, then its clasped expansion will have a clasp pointing from left to right (from C to D) if $A = D \multimap C$ and from right to left (again, from C to D) if $A = C \multimap D$, thereby showing the clasp in the opposite direction as it is written in the formula.

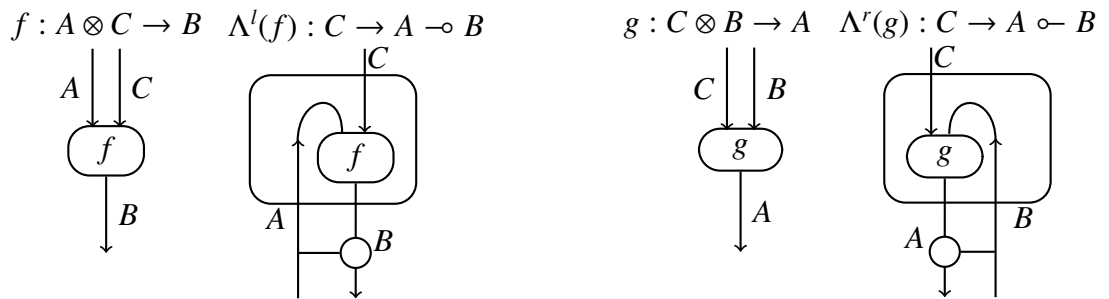


Figure 4.5: Diagrams for currying rules.

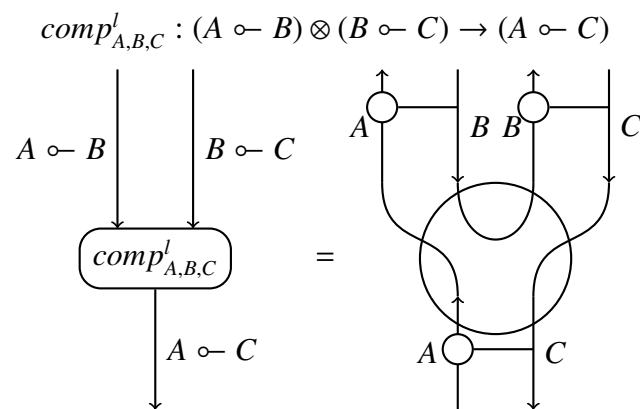
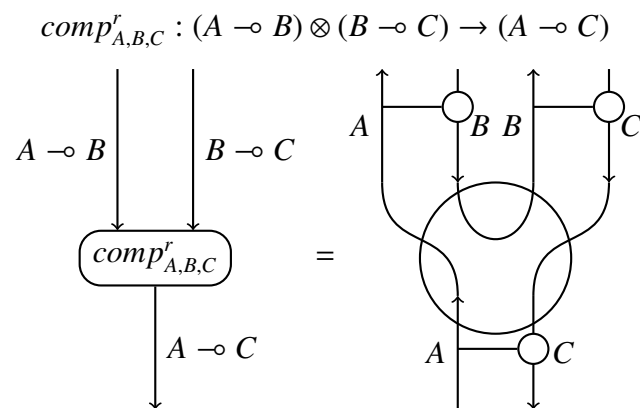


Figure 4.6: Diagrams for composition rules.

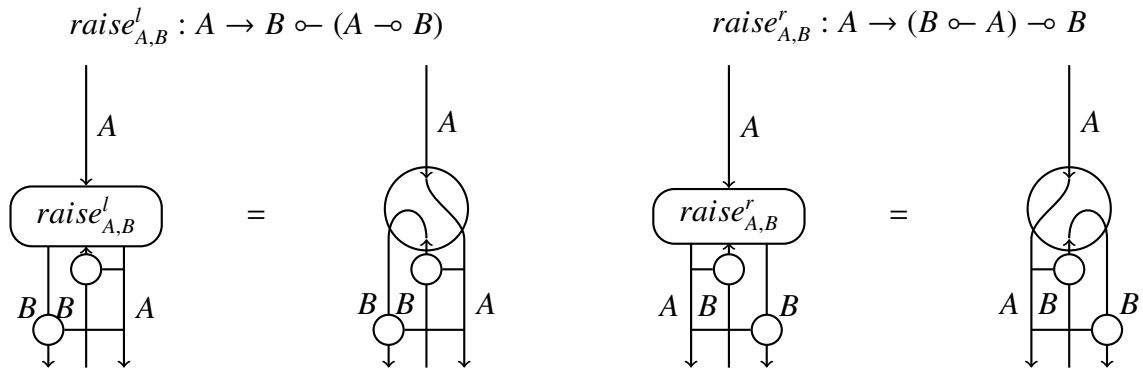


Figure 4.7: Diagrams for type raising rules.

To give a short example of how parses are represented diagrammatically, the diagrams for “men kill” and “men kill dogs” are shown in Figure 4.8. We assume nouns have the type N , intransitive verbs have type $N \multimap S$ and transitive verbs have the type $(N \multimap S) \multimap N$. The parse for “men kill” is just an $ev_{N,S}^l$. For the parse of “men kill dogs”, we start with one line with the type $N \multimap S$, do an $ev_{N \multimap S, N}^r$ with the object, then rewrite (marked with the dotted lines) this line to a clasp form with two lines of type N and S respectively, then do an $ev_{N,S}^l$ with the subject.

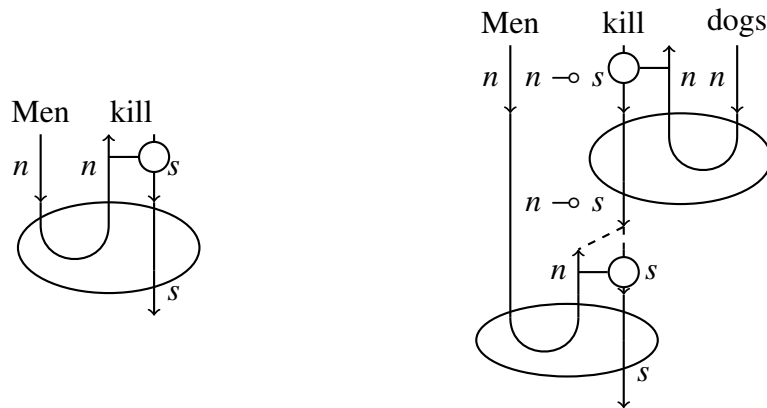


Figure 4.8: Diagrams for sample Lambek Grammar parses.

4.3.3 Defining a Functor

Equipped with this categorical representation \mathbf{LG} of a Lambek Grammar, we now turn to the task of defining a functorial passage from a category \mathbf{LG} to the category of vector spaces \mathbf{Vect} . Such a categorical passage was described in [17] which I co-authored with Mehrnoosh Sadrzadeh and Bob Coecke. I am indebted in particular to Mehrnoosh, who worked on the definition of this functor, and has permitted me to reproduce her work here

in my own words.

The bi-closed monoidal categories we used to represent \mathbf{LG} are structurally similar to the compact closed monoidal categories used to model pregroup categories P and \mathbf{FVect} , in that the evaluation morphisms ‘act’ like epsilon maps; for example:

$$ev_{A,B}^l : A \otimes (A \multimap B) \rightarrow B \cong \epsilon_A^r \otimes 1_B : A \otimes (A^r \otimes B) \rightarrow B$$

To generalise this similarity to a strict functor F , let each atomic object $A_{\mathbf{LG}}$ in $ob(\mathbf{LG})$ be assigned to a vector space $F(A_{\mathbf{LG}}) = A_{\mathbf{FVect}}$ in $ob(\mathbf{FVect})$. To simplify things notationally, we will drop the subscripts and just use the same letter. As a special case, the unit type I in \mathbf{LG} maps to the unit type in \mathbf{FVect} , namely \mathbb{R} , such that $F(I) = \mathbb{R}$.

The functorial passage for the monoidal tensor is simply the map of the monoidal tensor from \mathbf{LG} to \mathbf{FVect} , such that for any A and B in $ob(\mathbf{LG})$ we have:

$$F(A \otimes_{\mathbf{LG}} B) = F(A) \otimes_{\mathbf{FVect}} F(B)$$

Here too, we will dispense with indicating the subscript on the tensor operator.

Next comes the functorial definition for the \multimap and \multimap operations. Generally speaking, for the passage from a bi-closed monoidal category to a compact closed monoidal category, we would have the following two functorial passages:

$$F(A \multimap B) = F(A)^r \otimes F(B)$$

$$F(A \multimap B) = F(A) \otimes F(B)^l$$

However let us remember that in \mathbf{FVect} the adjoints are degenerate such that $A^l = A^r = A^*$ and $A^* \cong A$ for all vector spaces A , so we can do away with the adjoint notation. We therefore obtain a simpler functorial definition:

$$F(A \multimap B) = F(A) \otimes F(B)$$

$$F(A \multimap B) = F(A) \otimes F(B)$$

We now turn to the functorial passage for morphisms. The most complex case to consider here is slash introduction. Slash introduction states that for any morphism $f : A \otimes B \rightarrow C$ in $hom(\mathbf{LG})$ there are morphisms $\Lambda^l(f) : B \rightarrow A \multimap C$ and $\Lambda^r(f) : A \rightarrow C \multimap B$ in $hom(\mathbf{LG})$. Translating this to \mathbf{FVect} , this would entail that for any morphism $F(f) : F(A) \otimes F(B) \rightarrow F(C)$ in $hom(\mathbf{FVect})$ there must be $F(\Lambda^l(f)) : F(B) \rightarrow F(A) \otimes F(C)$ and $F(\Lambda^r(f)) : F(A) \rightarrow F(C) \otimes F(B)$ in $hom(\mathbf{FVect})$. For this passage to hold, we must show

that there are such curried morphisms for every such $F(f)$ in $\text{hom}(\mathbf{FVect})$.

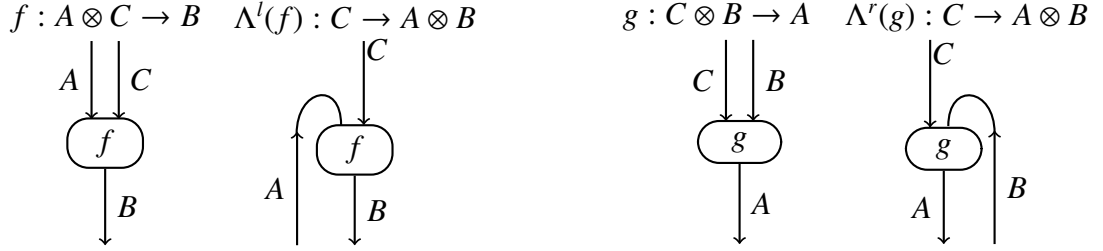


Figure 4.9: Diagrams for compact closed currying.

This is where a powerful feature of the graphical calculus for bi-closed categories of [3] comes into play: we can go from the diagrams of morphisms in bi-closed monoidal categories to those in compact closed categories simply by removing the bubbles and the clasps. This allows us to produce the diagrams for currying in \mathbf{FVect} simply by ‘cleaning up’ those for currying in \mathbf{LG} (or any other bi-closed monoidal category). I therefore show, in Figure 4.9, the diagrams for currying in a compact closed category, based on those shown earlier in Figure 4.5. From these diagrams, we can simply read the morphisms which are needed for currying rules to hold in \mathbf{FVect} , and these turn out to require nothing more than the $\eta_A : \mathbb{R} \rightarrow A \otimes A$ maps, which we represent as caps in the diagrammatic calculus, and which we know must exist in \mathbf{FVect} for any object A by virtue of the category being compact closed. This ‘magical’ use of the diagrams implicitly corresponds to the functorial passage from the bi-closed monoidal category representing the Lambek Grammar to a strict compact closed monoidal category which is free in its objects, and then passing from this category to \mathbf{FVect} using a similar functor as was defined for the passage from P to \mathbf{FVect} in §4.1.2. Using the diagrammatic calculus to avoid dealing with the underlying mathematics greatly simplifies the operation of defining a function in such a way, but it is important to be aware of the reliance on the freely generated nature of the source category \mathbf{LG} .

We can therefore read the following from these diagrams: for any morphism $F(f) : F(A) \otimes F(B) \rightarrow F(C)$ in $\text{hom}(\mathbf{FVect})$ there must be $F(\Lambda^l(f)) : B \rightarrow F(A) \otimes F(C)$ and $F(\Lambda^r(f)) : F(A) \rightarrow F(C) \otimes F(B)$ in $\text{hom}(\mathbf{FVect})$, where:

$$F(\Lambda^l(f)) = (1_A \otimes F(f)) \circ (\eta_{F(A)} \otimes 1_{F(B)}) : F(B) \rightarrow F(A) \otimes F(C)$$

$$F(\Lambda^r(f)) = (F(f) \otimes 1_{F(B)}) \circ (1_{F(A)} \otimes \eta_{F(B)}) : F(A) \rightarrow F(C) \otimes F(B)$$

So slash introduction is well defined under the functorial passage F .

The rest of the operations are far simpler to define, and can be read from Figures 4.4, 4.6

and 4.7:

$$F(\text{ev}_{A,B}^l) = \epsilon_{F(A)} \otimes 1_{F(B)} : F(A) \otimes F(A) \otimes F(B) \rightarrow F(B)$$

$$F(\text{ev}_{A,B}^r) = 1_{F(B)} \otimes \epsilon_{F(A)} : F(B) \otimes F(A) \otimes F(A) \rightarrow F(B)$$

$$F(\text{comp}_{A,B,C}^r) = 1_{F(A)} \otimes \epsilon_{F(B)} \otimes 1_{F(C)} : F(A) \otimes F(B) \otimes F(B) \otimes F(C) \rightarrow F(A) \otimes F(C)$$

$$F(\text{comp}_{A,B,C}^l) = 1_{F(A)} \otimes \epsilon_{F(B)} \otimes 1_{F(C)} : F(A) \otimes F(B) \otimes F(B) \otimes F(C) \rightarrow F(A) \otimes F(C)$$

$$F(\text{raise}_{A,B}^l) = (\eta_{F(B)} \otimes 1_{F(A)}) : F(A) \rightarrow F(B) \otimes F(B) \otimes F(A)$$

$$F(\text{raise}_{A,B}^r) = (1_{F(A)} \otimes \eta_{F(B)}) : F(A) \rightarrow F(A) \otimes F(B) \otimes F(B)$$

This completes the definition of a functorial passage between **LG** and **FVect**.

Chapter 5

Learning Procedures for a DisCoCat

Chapter Abstract

This chapter presents a procedure for generating concrete compositional distributional models of semantics from the DisCoCat framework presented in Chapter 3. It discusses several considerations as to the shape and structure of the semantic vector space for sentences, before presenting a learning procedure for the production of the semantic representation for words and relations based on corpus data.

Previously, in §3.3 I presented a categorical formalism which relates syntactic analysis steps to semantic composition operations. The structure of the syntactic representation used dictates the structure of the semantic spaces, but in exchange for this constraint, we are provided with natural composition functions by the syntactic analysis, rather than having to stipulate them *ad hoc*. Whereas the approaches to compositional DSMs presented in Chapter 2 either failed to take syntax into account during composition, or did so at the cost of not being able to compare sentences of different structure in a common space, this categorical approach projects all sentences into a common sentence space where they can be directly compared. However, this alone does not give us a compositional DSM.

As we have seen in the above examples, the structure of semantic spaces varies with syntactic types. We therefore cannot construct vectors for different syntactic types in the same way, as they live in spaces of different structure and dimensionality. Furthermore, nothing has yet been said about the structure of the sentence space S into which expressions reducing to type s are projected. If we wish to have a compositional DSM which leverages all the benefits of lexical DSMs and extends them to sentence-level distributional representations, we must specify a new sort of vector construction procedure.

In the original formulation of this formalism by [13, 19], examples of how such a compositional DSM could be used for logical evaluation are presented, where S is defined as a boolean space with True and False as basis vectors. However, the word vectors used are hand-written and specified model-theoretically, as the authors leave it for future research to determine how such vectors might be obtained from a corpus. In this chapter, I will discuss a new way of constructing vectors for Compositional DSMs, and of defining the sentence space S , in order to reconcile this powerful categorical formalism with the applicability and flexibility of standard distributional models.

5.1 Defining Sentence Space

5.1.1 Intuition

Assume the following sentences are all true:

1. The dogs chased the cats.
2. The dogs annoyed the cats.
3. The puppies followed the kittens.
4. The train left the station.
5. The president followed his agenda.

If asked which sentences have similar meaning, we would most likely point to the pair (1) and (3), and perhaps to a lesser degree (1) and (2), and (2) and (3). Sentences (4) and (5) obviously speak of a state of the world unrelated to that which is described by the other sentences.

If we compare these by truth value, we obviously have no means of making such distinctions. If we compare these by lexical similarity, (1) and (2) seem to be a closer match than (1) and (3). If we are classifying these sentences by some higher order relation such as ‘topic’, (5) might end up closer to (3) than (1). What, then, might cause us to pair (1) and (3)?

Intuitively, this similarity seems to be because the subjects and objects brought into relation by similar verbs are themselves similar. Abstracting away from tokens to some notion of property, we might say that both (1) and (3) express the fact that something furry and feline and furtive is being pursued by something aggressive (or playful) and canine. Playing along with the idea that lexical distributional semantics presents concepts (word

meanings) as ‘messy bundles of properties’, it seems only natural to have the way these properties are acted upon, qualified, and related as the basis for sentence-level distributional representations. In this respect, I here suggest that the sentence space S , instead of qualifying the truth value of a sentence, should express how the properties of the semantic objects within are qualified or brought into relation by verbs, adjectives, and other predicates and relations.

The above intuitions serve primarily to guide the development of a learning procedure as described below. The question of what are the “true semantics” of phrase and sentence vectors, of what they represent, is fundamental and difficult to answer. Such vectors could be interpreted as being the distributions of contexts in which sentences occur in an infinite idealised corpus; alternatively, they can be viewed simply as vectors of feature weights to be used in supervised machine learning algorithms such as classifiers, or as pre-training for unsupervised machine learning algorithms such as clustering algorithms. Either way, the construction procedure described below is general, and makes no commitments in terms of the interpretation of phrase and sentence vectors.

5.1.2 A Concrete Proposal

More specifically, I examine two suggestions for defining the sentence space, namely $S_I \cong N$ for sentences with intransitive verbs and $S_T \cong N \otimes N$ for sentences with transitive verbs. These definitions mean that the sentence space’s dimensions are commensurate with either those of N , or those of $N \otimes N$. These are by no means the only options, but as I will discuss here, they offer practical benefits.

In the case of S_I , the basis elements are labelled with unique basis elements of N , hence $\vec{s}_1 = \vec{n}_1$, $\vec{s}_2 = \vec{n}_2$, and so on. In the case of S_T , the basis elements are labelled with unique ordered pairs of elements from N , for example $\vec{s}_1 = \overrightarrow{(n_1, n_1)}$, $\vec{s}_2 = \overrightarrow{(n_2, n_1)}$, $\vec{s}_3 = \overrightarrow{(n_1, n_2)}$, and so on. Because of the isomorphism between S_T and $N \otimes N$, I will use the notations $\overrightarrow{(n_i, n_j)}$ and $\vec{n}_i \otimes \vec{n}_j$ interchangeably, as both constitute appropriate ways of representing the basis elements of such a space.

To propagate this distinction to the syntactic level, I define types s_I and s_T for intransitive and transitive sentences, respectively.

5.2 Noun-Oriented Types

5.2.1 Dealing with Nouns

Lambek’s pregroup types presented in [53] include a rich array of basic types and hand-designed compound types in order to capture specific grammatical properties. Here, for the sake of simplicity I will use a simpler set of grammatical types for experimental purposes, similar to some common types found in Combinatory Categorical Grammar (CCG) [77].

I assign a basic pregroup type n for all nouns, with an associated vector space N for their semantic representations. Furthermore, I will treat noun-phrases as nouns, assigning to them the same pregroup type and semantic space.

5.2.2 Dealing with Relational Words

CCG treats intransitive verbs as functions $NP \setminus S$ that consume a noun phrase and return a sentence, and transitive verbs as functions $(NP \setminus S) / NP$ that consume a noun phrase and return an intransitive verb function, which in turn consumes a noun phrase and returns a sentence. Using my distinction between intransitive and transitive sentences, I give intransitive verbs the type $n^r s_I$ associated with the semantic space $N \otimes S_I$, and transitive verbs the type $n^r s_T n^l$ associated with the semantic space $N \otimes S_T \otimes N$.

Adjectives, in CCG, are treated as functions NP / NP consuming a noun phrase and returning a noun phrase, and hence I give them the type nn^l and associated semantic space $N \otimes N$.

With the provision of a learning procedure for vectors in these semantic spaces, we can use these types to construct sentence vector representations for simple intransitive verb-based and transitive verb based sentences, with and without adjectives applied to subjects and objects.

5.3 Learning Procedures

5.3.1 Groundwork

To begin, I construct the semantic space N for all nouns in my lexicon (typically limited by the words available in the corpus used). Any distributional semantic model can be used for this stage, such as those presented in [21], or the lexical semantic models used by [63]. It seems reasonable to assume that higher quality lexical semantic vectors—as measured by metrics such as the WordSim353 test of [27]—will produce better relational vectors from

the procedure designed below. I will not test this hypothesis here, but note that it is an underlying assumption in most of the current literature on the subject [26, 63, 5].

Building upon the foundation of the thus-constructed noun vectors, I construct semantic representations for relational words. In pregroup grammars (or other combinatorial grammars such as CCG), we can view such words as functions taking as arguments those types present as adjoints in the compound pregroup type, and returning a syntactic object whose type is that of the corresponding reduction. For example, an adjective nn^l takes a noun or noun phrase n and returns a noun phrase n from the reduction $(nn^l)n \rightarrow n$. It can also compose with another adjective to return an adjective $(nn^l)(nn^l) \rightarrow nn^l$. I wish for my semantic representations to be viewed in the same way, such that the composition of an adjective with a noun $(1_N \otimes \epsilon_N)((N \otimes N) \otimes N)$ can be viewed as the application of a function $f : N \rightarrow N$ to its argument of type N .

To learn the representations of such functions, I assume that their meaning can be characterised by the properties that their arguments hold in the corpus, rather than just by their context as is the case in lexical distributional semantic models. To give an example, rather than learning what the adjective “angry” means by observing that it co-occurs with words such as “fighting”, “aggressive” or “mean”, we can learn its meaning by observing that it typically takes, as argument, words that *co-occur with* words such as “fighting”, “aggressive” and “mean”. While in the lexical semantic case, such associations might only rarely occur in the corpus, in this indirect method we learn what properties the adjective relates to even if they do not co-occur with it directly.

In turn, through composition with its argument, I expect the function for such an adjective to *strengthen* the properties that characterise it in the representation of the object it takes as argument. Let us assume “angry” is characterised by arguments that have high basis weights for basis elements corresponding to the concepts (or context words) “fighting”, “aggressive” and “mean”, and relatively low counts for semantically different concepts such as “passive”, “peaceful” and “loves”. When I apply “angry” to “dog” the vector for the compound “angry dog” should contain some of the information found in the vector for “dog”. But this vector should also have *higher* values for the basis weights of “fighting”, “aggressive” and “mean”, and correspondingly lower values for the basis weights of “passive”, “peaceful”, “loves”.

5.3.2 A Learning Algorithm

To turn this idea into a concrete algorithm for constructing the semantic representation for relations of any arity, as first presented in [38], let’s examine how we would deal with this

for binary relations such as transitive verbs. If a transitive verb of semantic type $N \otimes S_T \otimes N$ is viewed as a function $f : N \times N \rightarrow S_T$ which expresses the extent to which the properties of subject and object are brought into relation by the verb, we learn the meaning of the verb by looking at what properties are brought into relation by the verb in terms of what arguments it takes in a corpus. Recall that the vector for a verb v , $\vec{v} \in N \otimes S_T \otimes N$, can be expressed as the weighted superposition of its basis elements:

$$\vec{v} = \sum_{ijk} c_{ijk}^v \vec{n}_i \otimes \vec{s}_j \otimes \vec{n}_k$$

I take the set of vectors for the subject and object of v in the corpus to be the set of pairs $arg_v = \{(\overrightarrow{SUBJ}_l, \overrightarrow{OBJ}_l)\}_l$. I wish to calculate the basis weightings $\{c_{ijk}^v\}_{ijk}$ for v . Exploiting my earlier definition of the basis $\{s_j\}_j$ of S_T which states that for any value of i and k there is some value of j such that $s_j = (n_i, n_k)$, I define $\Delta_{ijk} = 1$ if $s_j = (n_i, n_k)$ and 0 otherwise. Using all of the above, I define the calculation of each basis weight c_{ijk}^v as:

$$c_{ijk}^v = \sum_l \Delta_{ijk} c_i^{SUBJ_l} c_k^{OBJ_l}$$

This allows for a full formulation of \vec{v} as follows:

$$\vec{v} = \sum_l \sum_{ijk} \Delta_{ijk} c_i^{SUBJ_l} c_k^{OBJ_l} \vec{n}_i \otimes \vec{s}_j \otimes \vec{n}_k$$

The nested sums here may seem computationally inefficient, seeing how this would involve computing $size(arg_v) \times dim(N)^2 \times dim(S) = size(arg_v) \times dim(N)^4$ products. However, using the decomposition of basis elements of S into pairs of basis elements of N (effectively basis elements of $N \otimes N$), we can remove the Δ_{ijk} term and ignore all values of j where $s_j \neq (n_i, n_k)$, since the basis weight for this combination of indices would be 0. Therefore I simplify the formulation of \vec{v} :

$$\vec{v} = \sum_l \sum_{ik} c_i^{SUBJ_l} c_k^{OBJ_l} \vec{n}_i \otimes \overrightarrow{(n_i, n_k)} \otimes \vec{n}_k$$

This representation is still bloated: we perform less calculations, but still obtain a vector in which all the basis weights where $s_j \neq (n_i, n_k)$ are 0, hence where only $dim(N)^2$ of the $dim(N)^4$ values are non-zero. In short, the vector weights for \vec{v} are, under this learning algorithm, entirely characterised by the values of a $dim(N)$ by $dim(N)$ matrix, the entries

of which are products $c_i^{SUBJ_l} c_k^{OBJ_l}$ where i and k have become row and column indices.

Using this and my definition of S_T as a space isomorphic to $N \otimes N$, I can formulate a compact, ‘reduced’ expression of \vec{v} as follows. Let the Kronecker product of two vectors $\vec{u}, \vec{w} \in N$, written $\vec{u} \otimes \vec{w} \in N \otimes N$, be as follows:

$$\vec{u} \otimes \vec{w} = \sum_{ij} c_i^u c_j^w \vec{n}_i \otimes \vec{n}_j$$

Equipped with this definition, I can formulate the compact form of \vec{v} :

$$\begin{aligned} \vec{v} &= \sum_l \sum_{ik} c_i^{SUBJ_l} c_k^{OBJ_l} \vec{n}_i \otimes \vec{n}_k \\ &= \sum_l \overrightarrow{SUBJ_l} \otimes \overrightarrow{OBJ_l} \end{aligned}$$

In short, we are only required to iterate through the corpus once, taking for each instance of a transitive verb v the Kronecker product of its subject and object, and summing these across all instances of v . It is simple to see that no information was discarded relative to the previous definition of \vec{v} : the dimensionality reduction by a factor of $\dim(N)^2$ simply discards all basis elements for which the basis weight was 0 by default.

5.3.3 Problems with Reduced Representations

This raises a small problem though: this compact representation can no longer be used in the compositional mechanism presented in §3.3, as the dimensions of \vec{v} no longer match those which it is required to have according to its syntactic type. However, a solution can be devised if we return to the sample calculation, shown in §3.3.3, of the composition of a transitive verb with its arguments. The composition is as follows:

$$(\epsilon_N \otimes 1_S \otimes \epsilon_N)(\overrightarrow{SUBJ} \otimes \vec{v} \otimes \overrightarrow{OBJ}) = \sum_{ikm} c_i^{SUBJ} c_{ikm}^v c_m^{OBJ} \vec{s}_k$$

where the verb v is represented in its non-compact form. By introducing the compact representation permitted by the isomorphism $S_T \cong N \otimes N$ I can express this as

$$\overrightarrow{SUBJ \ v \ OBJ} = \sum_{im} c_i^{SUBJ} c_{im}^v c_m^{OBJ} \vec{n}_i \otimes \vec{n}_m$$

where v is represented in its compact form. Furthermore, by introducing the component wise multiplication operation \odot :

$$\vec{u} \odot \vec{v} = \sum_i c_i^u c_i^v \vec{n}_i$$

I can show the general form of transitive verb composition using the reduced verb representation to be as follows:

$$\begin{aligned} \overrightarrow{SUBJ v OBJ} &= \sum_{im} c_i^{SUBJ} c_{im}^v c_m^{OBJ} \vec{n}_i \otimes \vec{n}_m \\ &= \left(\sum_{im} c_{im}^v \vec{n}_i \otimes \vec{n}_m \right) \odot \left(\sum_{im} c_i^{SUBJ} c_m^{OBJ} \vec{n}_i \otimes \vec{n}_m \right) \\ &= \vec{v} \odot \left(\overrightarrow{SUBJ} \otimes \overrightarrow{OBJ} \right) \end{aligned}$$

Considering the diagrammatic form of reduced representations may help understand their construction and how composition works. Let \vec{v} be the reduced representation of a verb, and the function f^v be defined as follows:

$$f^v : N \otimes N \rightarrow N \otimes N :: \vec{a} \otimes \vec{b} \mapsto \vec{v} \odot (\vec{a} \otimes \vec{b})$$

The diagrammatic representation of a verb is shown in Figure 5.1 as being embedded within the full representation of the verb. Composition of a subject and object with the full representation results in a yank which feeds the vectors to the function f^v which applies the reduced representation to the Kronecker product of the subject and object vectors, as shown in Figure 5.2. In sum, I am doing exactly the same sort of computation when I use reduced representations as I do when I used full representations. The ‘simplification’ of reduced representations is only a reduction in the cost of the computation; in all other respects, it is identical to computing using ϵ maps.

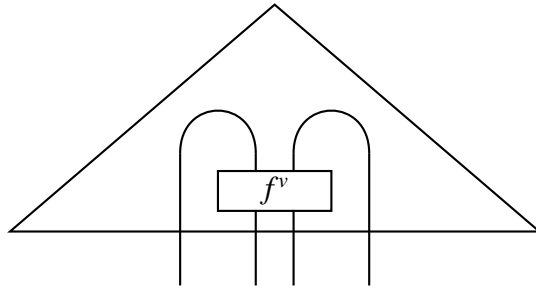


Figure 5.1: Diagrammatic form of reduced representations.

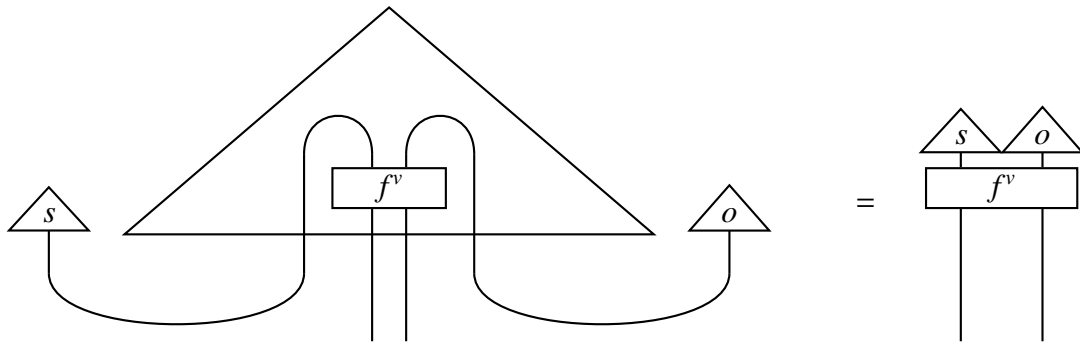


Figure 5.2: Composition under reduced representations.

To summarise what I have done with transitive verbs:

1. I have treated them as functions taking two nouns and returning a sentence in a space $S_T \cong N \otimes N$.
2. I have built the semantic representations for these functions by counting which properties of subject and object noun vectors are related by the verb in transitive sentences in a corpus.
3. I have assumed that output properties are a function of input properties by the use of the Δ_{ijk} function, i.e. the weights associated to n_i from the subject argument and with n_k from the object argument only affect the ‘output’ weight for the sentence basis element $\vec{s}_j = \vec{n}_i \otimes \vec{n}_k$.
4. I have shown that this leads to a compact representation of the verb’s semantic form, and an optimised learning procedure.
5. I have shown that the composition operations of this formalism can be adapted to this compact representation.

The compact representation and amended composition operation hinge on the choice of $S_T \cong N \otimes N$ as output type for $N \otimes N$ as input type (the pair of arguments the verb takes), justifying my choice of a transitive sentence space. In the intransitive case, the same phenomenon can be observed, since such a verb takes as argument a vector in N and produces a vector in $S_I \cong N$. Furthermore, my choice to make all other types dependent on one base type—namely n (with associated semantic space N)—yields the same property for every relational word we wish to learn: the output type is the same as the concatenated (on the syntactic level) or tensored (on the semantic level) input types. It is this symmetry between input and output types that guarantees that any m -ary relation, expressed in the

original formulation as an element of tensor space $\underbrace{N \otimes \dots \otimes N}_{2m}$ has a compact representation in $\underbrace{N \otimes \dots \otimes N}_m$ where the i th basis weight of the reduced representation stands for the degree to which the i th element of the input vector affects the i th element of the output vector.

5.3.4 A Generalised Algorithm for Reduced Representations

The learning procedure discussed above allows the specification of a generalised learning algorithm for reduced representations, first presented in [36], which is as follows. Each relational word P (i.e. those words with compound pregroup types) with grammatical type π and m adjoint types $\alpha_1, \alpha_2, \dots, \alpha_m$ is encoded as an $(r \times \dots \times r)$ multi-dimensional array with m degrees of freedom (i.e. an order m tensor). Since the vector space N has a fixed basis, each such array is represented in vector form as follows:

$$\vec{P} = \sum_{\underbrace{ij \dots \zeta}_m} c_{ij \dots \zeta} \underbrace{(\vec{n}_i \otimes \vec{n}_j \otimes \dots \otimes \vec{n}_\zeta)}_m$$

This vector lives in the tensor space $\underbrace{N \otimes N \otimes \dots \otimes N}_m$. Each $c_{ij \dots \zeta}$ is computed according to the procedure described in Figure 5.3.

- 1) Consider a sequence of words containing a relational word 'P' and its arguments w_1, w_2, \dots, w_m , occurring in the same order as described in P's grammatical type π . Refer to these sequences as 'P'-relations. Suppose there are K of them.
- 2) Retrieve the vector \vec{w}_l of each argument w_l .
- 3) Suppose w_1 has weight c_i^1 on basis vector \vec{n}_i , w_2 has weight c_j^2 on basis vector \vec{n}_j, \dots , and w_m has weight c_ζ^m on basis vector \vec{n}_ζ . Multiply these weights

$$c_i^1 \times c_j^2 \times \dots \times c_\zeta^m$$

- 4) Repeat the above steps for all the K 'P'-relations, and sum the corresponding weights

$$c_{ij \dots \zeta} = \sum_{k=1}^K (c_i^1 \times c_j^2 \times \dots \times c_\zeta^m)_k$$

Figure 5.3: Procedure for learning weights for matrices of words 'P' with relational types π of m arguments.

Linear algebraically, this procedure corresponds to computing the following sum:

$$\vec{P} = \sum_k (\vec{w}_1 \otimes \vec{w}_2 \otimes \dots \otimes \vec{w}_m)_k$$

The general formulation of composing a relational word P with its arguments arg_1, \dots, arg_m is now expressed as

$$\vec{P} \odot (\overrightarrow{arg_1} \otimes \dots \otimes \overrightarrow{arg_m})$$

For example the computation of “furry cats nag angry dogs” would correspond to the following operation:

$$\overrightarrow{\text{furry cats nag angry dogs}} = \overrightarrow{\text{nag}} \odot \left(\left(\overrightarrow{\text{furry}} \odot \overrightarrow{\text{cat}} \right) \otimes \left(\overrightarrow{\text{angry}} \odot \overrightarrow{\text{dog}} \right) \right)$$

5.3.5 Example

To give an example, taken from [36], I demonstrate how the meaning of the word ‘show’ might be learned from a corpus and then composed. Suppose there are two instances of the verb ‘show’ in the corpus:

- s_1 = table show result
- s_2 = map show location

The vector of ‘show’ is

$$\overrightarrow{\text{show}} = \overrightarrow{\text{table}} \otimes \overrightarrow{\text{result}} + \overrightarrow{\text{map}} \otimes \overrightarrow{\text{location}}$$

Consider a vector space N with four basis vectors ‘far’, ‘room’, ‘scientific’, and ‘elect’. The TF/IDF-weighted values for vectors of selected nouns (built from the British National Corpus) are as shown in Table 5.1.

i	\vec{n}_i	table	map	result	location	master	dog
1	far	6.6	5.6	7	5.9	4.3	5.5
2	room	27	7.4	1.0	7.3	9.1	7.6
3	scientific	0	5.4	13	6.1	4.1	0
4	elect	0	0	4.2	0	6.2	0

Table 5.1: Sample weights for selected noun vectors.

Part of the matrix compact representation of ‘show’ is presented in Table 5.2.

As a sample computation, the weight c_{11} for basis element (\vec{n}_1, \vec{n}_1) , i.e. $(\overrightarrow{\text{far}}, \overrightarrow{\text{far}})$, is computed by multiplying weights of ‘table’ and ‘result’ on $\overrightarrow{\text{far}}$, i.e. 6.6×7 , multiplying weights of ‘map’ and ‘location’ on $\overrightarrow{\text{far}}$, i.e. 5.6×5.9 then adding these $46.2 + 33.04$ and obtaining the total weight 79.24.

I now wish to compute the vector for the sentence “[the] master shows [his] dog” (omitting the determiner and possessive for simplicity). The calculation will be:

	far	room	scientific	elect
far	79.24	47.41	119.96	27.72
room	232.66	80.75	396.14	113.2
scientific	32.94	31.86	32.94	0
elect	0	0	0	0

Table 5.2: Sample semantic matrix for ‘show’.

$$\begin{aligned}
& \overrightarrow{\text{master show dog}} \\
&= \overrightarrow{\text{show}} \odot (\overrightarrow{\text{master}} \otimes \overrightarrow{\text{dog}}) \\
&= \begin{bmatrix} 79.24 & 47.41 & 119.96 & 27.72 \\ 232.66 & 80.75 & 396.14 & 113.2 \\ 32.94 & 31.86 & 32.94 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \odot \begin{bmatrix} 23.65 & 32.68 & 0 & 0 \\ 50.05 & 69.16 & 0 & 0 \\ 22.55 & 31.16 & 0 & 0 \\ 34.1 & 47.12 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 1874.03 & 1549.36 & 0 & 0 \\ 11644.63 & 5584.67 & 0 & 0 \\ 742.80 & 992.76 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}
\end{aligned}$$

The row-wise flattening of the final matrix representation gives us the result we seek, namely the sentence vector in S_T for “[the] master shows [his] dog”:

$$\overrightarrow{\text{master show dog}} = [1874, 1549, 0, 0, 11645, 5585, 0, 0, 743, 993, 0, 0, 0, 0, 0]$$

5.3.6 An Efficient Alternative

To complete this chapter, let us consider an alternative and efficient way of learning the representations for transitive verbs, which I will call the **Kronecker** method. I first presented this efficient learning method with Mehrnoosh Sadrzadeh in [37], where we observed that the compact representation of a verb in the DisCoCat framework, under the assumptions presented earlier in this chapter, can be viewed as $\dim(N) \times \dim(N)$ matrices in $N \otimes N$. We considered alternatives to the algorithm presented earlier for the construction of such matrices, and were surprised by the results of the **Kronecker** method, wherein we replaced the matrix learned by our algorithm with the Kronecker product of the lexical semantic vectors for the verb. Concretely this means that given the lexical vector $\overrightarrow{v_{lex}}$ for a transitive

verb, built as we would build noun vectors, we construct the matrix representation **verb** of the verb as follows:

$$\mathbf{verb} = \vec{v}_{lex} \otimes \vec{v}_{lex}$$

Further analysis performed since the publication of that paper can help to understand why this method might work. Using the following property that for any vectors $\vec{a}, \vec{b}, \vec{c}, \vec{d}$, we have

$$(\vec{a} \otimes \vec{b}) \odot (\vec{c} \otimes \vec{d}) = (\vec{a} \odot \vec{c}) \otimes (\vec{b} \odot \vec{d})$$

we can see that the **Kronecker** model's composition operation can be expressed as

$$\mathbf{Kronecker} : \overrightarrow{\text{subject verb object}} = \left(\overrightarrow{\text{verb}} \odot \overrightarrow{\text{subject}} \right) \otimes \left(\overrightarrow{\text{verb}} \odot \overrightarrow{\text{object}} \right)$$

Bearing in mind that the cosine measure we are using as a similarity metric is equivalent to the inner product of two vectors normalised by the product of their length

$$\text{cosine}(\vec{a}, \vec{b}) = \frac{\langle \vec{a} | \vec{b} \rangle}{\|\vec{a}\| \times \|\vec{b}\|}$$

and the following property of the inner product of Kronecker products

$$\langle \vec{a} \otimes \vec{b} | \vec{c} \otimes \vec{d} \rangle = \langle \vec{a} | \vec{c} \rangle \times \langle \vec{b} | \vec{d} \rangle$$

we finally observe that comparing two sentences under **Kronecker** corresponds to the following computation:

$$\begin{aligned} & \text{cosine}(\overrightarrow{\text{subject verb}_1 \text{ object}}, \overrightarrow{\text{subject verb}_2 \text{ object}}) \\ &= \alpha \left\langle \left(\overrightarrow{\text{verb}_1} \otimes \overrightarrow{\text{verb}_1} \right) \odot \left(\overrightarrow{\text{subject}} \otimes \overrightarrow{\text{object}} \right) \middle| \left(\overrightarrow{\text{verb}_2} \otimes \overrightarrow{\text{verb}_2} \right) \odot \left(\overrightarrow{\text{subject}} \otimes \overrightarrow{\text{object}} \right) \right\rangle \\ &= \alpha \left\langle \left(\overrightarrow{\text{verb}_1} \odot \overrightarrow{\text{subject}} \right) \otimes \left(\overrightarrow{\text{verb}_1} \odot \overrightarrow{\text{object}} \right) \middle| \left(\overrightarrow{\text{verb}_2} \odot \overrightarrow{\text{subject}} \right) \otimes \left(\overrightarrow{\text{verb}_2} \odot \overrightarrow{\text{object}} \right) \right\rangle \\ &= \alpha \left\langle \left(\overrightarrow{\text{verb}_1} \odot \overrightarrow{\text{subject}} \right) \middle| \left(\overrightarrow{\text{verb}_2} \odot \overrightarrow{\text{subject}} \right) \right\rangle \left\langle \left(\overrightarrow{\text{verb}_1} \odot \overrightarrow{\text{object}} \right) \middle| \left(\overrightarrow{\text{verb}_2} \odot \overrightarrow{\text{object}} \right) \right\rangle \end{aligned}$$

where α is the normalisation factor

$$\alpha = \frac{1}{\|\overrightarrow{\text{subject verb}_1 \text{ object}}\| \times \|\overrightarrow{\text{subject verb}_2 \text{ object}}\|}$$

I note here that the **Kronecker** model is effectively a parallel application of the multiplicative model presented in §2.3.2, combining the subject and verb, and object and verb

separately. In short, it more or less constitutes the introduction of some mild syntactic sensitivity into the multiplicative model of [63].

I can generalise this Kronecker model to any relation of any arity, provided that it takes n inputs and produces n outputs of matching types (since all I am doing is component-wise multiplication of reduced representations). The general learning procedure for learning an n -ary relation using the Kronecker method is as follows:

1. We learn the lexical vector R for the relation.
2. We produce its tensor representation T^R by computing

$$T^R = \bigotimes_{i=1}^n R$$

3. To compose T^R with its N arguments $\{\vec{a}_1, \dots, \vec{a}_n\}$ we compute:

$$T^R \odot \left(\bigotimes_{i=1}^n \vec{a}_i \right)$$

This is, in some sense, a Kronecker product-based generalisation of the multiplicative model of [63], as the case for $n = 1$ is simply component-wise multiplication.

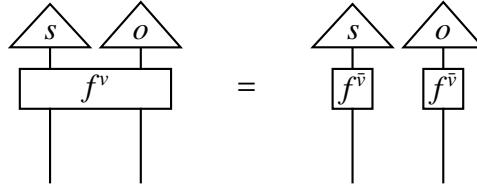


Figure 5.4: Composition under the Kronecker model.

To conclude this section, let us examine the diagrammatic representation of composition under the Kronecker model. We saw, in Figure 5.2, that reduced representations could be seen as being diagrammatically embedded inside full representations. Furthermore, these diagrams can be simplified through yank operations to show the application of some function f^v to the Kronecker product of subject and object, where f^v corresponds to the component-wise multiplication of Kronecker product two arguments with the reduced representation of the verb. In the case of the Kronecker model, this reduced representation is itself the Kronecker product of the lexical vectors for the verb, which I will call \bar{v} . Let a function $f^{\bar{v}}$ be defined as follows:

$$f^{\bar{v}} : N \rightarrow N :: \vec{a} \mapsto \vec{a} \odot \bar{v}$$

I can therefore express f^v in terms of $f^{\bar{v}}$ as follows:

$$f^v = f^{\bar{v}} \otimes f^{\bar{v}}$$

which allows us to formulate the diagrammatic representation of composition under the Kronecker model as shown in Figure 5.4. The general form of the Kronecker model can be shown in a similar way. If we construct the representation of some n -ary relation R according to the Kronecker model, by taking the Kronecker product of its lexical vector \bar{R} n times, then we have:

$$f^R = \bigotimes_{i=1}^n f^{\bar{R}}$$

and the composition of R with its n arguments $\{\vec{a}_1, \dots, \vec{a}_n\}$ can be represented diagrammatically as shown in Figure 5.5.

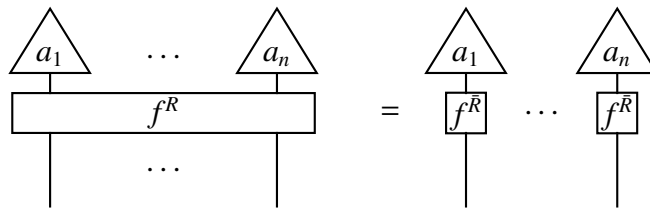


Figure 5.5: Composition under the generalised Kronecker model.

Part III

Practice

Chapter 6

Evaluating a DisCoCat

Chapter Abstract

This chapter presents three phrase-similarity detection experiments designed to evaluate the models discussed in Chapter 5, and compare them to selected other models discussed in Chapter 2 within the context of these experiments. It shows the concrete models generated by the DisCoCat framework to outperform other competing models.

Evaluating compositional models of semantics is no easy task: first, we are trying to evaluate how well the compositional process works; second, we also are trying to determine how useful the final representation—the *output* of the composition—is, relative to our needs.

The scope of this second problem covers most phrase and sentence-level semantic models, from bag-of-words approaches in information retrieval to logic-based formal semantic models, via language models used for machine translation. It is heavily task dependent, in that a representation that is suitable for machine translation may not be appropriate for textual inference tasks, and one that is appropriate for information retrieval may not be ideal for paraphrase detection. Therefore this aspect of semantic model evaluation ideally should take the form of application-oriented testing, e.g. to test semantic representations designed for machine translation purposes, we should use a machine translation evaluation task.

The DisCoCat framework of [13, 19], described above in Chapter 3, allows for the composition of any words of any syntactic type. The general learning algorithm presented in §5.3 technically can be applied to learn and model relations of any semantic type. However, many open questions remain, such as how to deal with logical words, determiners and quantification, and how to reconcile the different semantic types used for sentences with

transitive and intransitive sentences. I will leave these questions for future work, briefly discussing some of them in Chapter 7. In the meantime, we concretely are left with a way of satisfactorily modelling only simple sentences without having to answer these bigger questions.

With this in mind, in this chapter I present a series of experiments centred around evaluating how well various models of semantic vector composition (along with the one described in Chapter 5) perform in a phrase similarity comparison task. This task aims to test the quality of a compositional process by determining how well it forms a clear joint meaning for potentially ambiguous words. The intuition here is that tokens, on their own, can have several meanings, and that it is through the compositional process—through giving them *context*—that we understand their specific meaning. For example, “bank” itself could (amongst other meanings) mean a river bank or a financial bank; yet in the context of a sentence such as “The bank refunded the deposit” it is likely we are talking about the financial institution.

I will present three datasets designed to evaluate how well word-sense disambiguation occurs as a by-product of composition. I begin by describing the first dataset, based around noun-intransitive verb phrases, in §6.1. In §6.2, I present a dataset based around short transitive-verb phrases (a transitive verb with subject and object). In §6.3, I discuss a new dataset, based around short transitive-verb phrases where the subject and object are qualified by adjectives. I leave discussion of these results for §6.4.

6.1 First Experiment

This first experiment, which is based on that of Mitchell and Lapata [63], and which I presented in [36] with Mehrnoosh Sadrzadeh, evaluates the degree to which an ambiguous intransitive verb (e.g. “draws”) is disambiguated by combination with its subject.

6.1.1 Dataset Description

The dataset¹ comprises 120 pairs of intransitive sentences, each of the form “NOUN VERB”. These sentence pairs are generated according the following procedure, which will be the basis for the construction of the other datasets discussed below:

1. A number of ambiguous intransitive verbs (15, in this case) are selected from frequently occurring verbs in the corpus.

¹Available at <http://homepages.inf.ed.ac.uk/s0453356/results>.

2. For each verb V , two mutually exclusive synonyms V_1 and V_2 of the verb are produced, and each is paired with the original verb separately (for a total of 30 verb pairs). These are generated by taking maximally distant pairs of synonyms of the verb on Wordnet (e.g.), but any method could be used here.
3. For each *pair of verb pairs* (V, V_1) and (V, V_2) , two frequently occurring nouns N_1 and N_2 are picked from the corpus, one for each synonym of V . For example if V is “glow” and the synonyms V_1 and V_2 are “beam” and “burn”, we might choose “face” as N_1 because a face glowing and a face beaming mean roughly the same thing, and “fire” as N_2 because a fire glowing and a fire burning mean roughly the same thing.
4. By combining the nouns with the verb pairs, we form two high similarity triplets (V, V_1, N_1) and (V, V_2, N_2) , and two low similarity triplets (V, V_1, N_2) and (V, V_2, N_1) .

The last two steps can be repeated to form more than four triplets per pair of verb pairs. In [63], eight triplets are generated for each pair of verb pairs, obtaining a total of 120 triplets from the 15 original verbs. Each triplet, along with its HIGH or LOW classification (based on the choice of noun for the verb pair) is an entry in the dataset, and can be read as a pair of sentences: (V, V_i, N) translates into the intransitive sentences “ $N V$ ” and “ $N V_i$ ”.

Finally, the dataset is presented, without the HIGH/LOW ratings, to human annotators. These annotators are asked to rate the similarity of meaning of the pairs of sentences in each entry on a scale of 1 (low similarity) to 7 (high similarity). The final form of the dataset is a set of lines each containing:

- A (V, V_i, N) triplet
- A HIGH or LOW label for that triplet
- An annotator identifier and the annotator’s score for that triplet

Sample sentences from this dataset are shown in Table 6.1.

Sentence 1	Sentence 2
butler bow	butler submit
head bow	head stoop
company bow	company submit
government bow	government stoop

Table 6.1: Example entries from the intransitive dataset without annotator score, first experiment.

6.1.2 Evaluation Methodology

This dataset is used to compare various compositional distributional semantic models according to the following procedure:

1. For each entry in the dataset, the representation of the two sentences “ $N V$ ” and “ $N V_i$ ”, which we will name S_1 and S_2 , formed from the entry triple (V, V_i, N) is constructed by the model.
2. The similarity of these sentences according to the model’s semantic distance measure constitutes the *model score* for the entry.
3. The rank correlation of entry model scores against entry annotator scores is calculated using Spearman’s rank correlation coefficient ρ .

The Spearman ρ scores are values between -1 (inverse rank correlation) and 1 (perfect correlation). The higher the ρ score, the higher the compositional model can be said to produce sentence representations that match human understanding of sentence meaning, when it comes to comparing the meaning of sentences. As such, I will rank the models evaluated using the task by decreasing order of ρ score.

One of the principal appealing features of Spearman’s ρ is that the coefficient is rank based: it does not require models’ semantic similarity metrics to be normalised for a comparison to be made. One consequence is that a model providing excellent rank correlation with human scores, but producing model scores on a small scale (e.g. values between 0.5 and 0.6) will obtain a higher ρ score than a model producing models scores on a larger scale (e.g. between 0 and 1) but with less perfect rank correlation. If we wished to then use the former model in a task requiring some greater degree of numerical separation (let’s say 0 for non-similar sentences and 1 for completely similar sentences), we could simply renormalise the model scores to fit the scale. By eschewing score normalisation as an evaluation factor, we minimise the risk of erroneously ranking one model over another.

Finally, in addition to computing the rank alignment coefficient between model scores and annotator scores, [63] calculate the mean models scores for entries labelled HIGH, and for entries labelled LOW. This information is reported in their paper as additional means for model comparison. However, for the same reason I considered Spearman’s ρ to be a fair means of model comparison, namely in that it required no model score normalisation procedure and thus was less likely to introduce error by adding such a degree of freedom, we consider the HIGH/LOW means to be inadequate grounds for comparison precisely because it *requires* normalised model scores for comparison to be meaningful. As such, I

will not include these mean scores in the presentation of this experiment’s results, or any further experiments in this Chapter.

6.1.3 Models Compared

In this experiment, I compare the best and worst performing models of [63] to my own. I begin by building a vector space W for all words in the corpus, using standard distributional semantic model construction procedures (n -word window) with the parameters of [63]. Specifically, the basis elements of this vector space are the 2000 most frequently occurring words in the British National Corpus (BNC), excluding common stop words. For my evaluation, the corpus was lemmatised using Carroll’s Morpha [62], which was applied as a by-product of my parsing of the BNC with the C&Ctools parser of [20].

The context of each occurrence of a word in the corpus was defined to be five words on either side. After the vector for each word w was produced, the basis weights c_i^w associated with context word b_i were normalised by the following ratio of probabilities weighting scheme:

$$c_i^w = \frac{P(b_i|w)}{P(b_i)} = \frac{f_{w \wedge b_i} f_{b_i}}{f_{b_i} f_w}$$

where f_{b_i} is the total number of context word in the corpus.

Let \overrightarrow{verb} and \overrightarrow{noun} be the lexical semantic vectors for the verb and the noun, from W . It should be evident that there is no significant difference in using W for nouns in lieu of building a separate vector space N strictly for noun vectors.

As a first baseline, **Verb Baseline**, I ignored the information provided by the noun in constructing the sentence representation, effectively comparing the semantic content of the verbs:

$$\mathbf{Verb\ Baseline} : \overrightarrow{noun\ verb} = \overrightarrow{verb}$$

The models from [63] which I evaluate here are those which were strictly unsupervised (i.e. no free parameters for composition). These are the additive model **Add**, wherein

$$\mathbf{Add} : \overrightarrow{noun\ verb} = \overrightarrow{noun} + \overrightarrow{verb}$$

and the multiplicative model **Multiply**, wherein

$$\mathbf{Multiply} : \overrightarrow{noun\ verb} = \overrightarrow{noun} \odot \overrightarrow{verb}$$

Other models with parameters which must be optimised against a held-out section the dataset are presented in [63]. I omitted them here principally because they do not per-

form as well as **Multiply**, but also because the need to optimise the free parameters for this experiment and other datasets makes comparison with completely unsupervised models more difficult, and less fair.

I also evaluate the **Categorical** model from Chapter 5 here, wherein

$$\mathbf{Categorical} : \overrightarrow{noun\ verb} = \overrightarrow{verb_{cat}} \odot \overrightarrow{noun}$$

where $\overrightarrow{verb_{cat}}$ is the compact representation of the relation the verb stands for, computed according to the procedure described in Chapter 5. It should be noted that for the case of intransitive verbs, this composition operation is mathematically equivalent to that of the **Multiply** model (as component-wise multiplication \odot is a commutative operation), the difference being the learning procedure for the verb vector.

For all such vector-based models, the similarity of two sentences s_1 and s_2 is taken to be the cosine similarity of their vectors, defined in §2.2:

$$similarity(s_1, s_2) = cosine(\overrightarrow{s_1}, \overrightarrow{s_2})$$

In addition to these vector-based methods, I define an additional baseline and an upper-bound. The additional baseline, **Bigram Baseline**, is a bigram-based language model trained on the BNC with SRILM [79], using the standard language model settings for computing log-probabilities of bigrams. To determine the semantic similarity of sentences $s_1 = noun\ verb_1$ and $s_2 = noun\ verb_2$ I assumed sentences have mutually conditionally independent properties, and computed the joint probability:

$$similarity(s_1, s_2) = \log P(s_1 \wedge s_2) = \log(P(s_1)P(s_2)) = \log P(s_1) + \log P(s_2)$$

The upper bound of the dataset, **UpperBound**, was taken to be the inter-annotator agreement: the average of how each annotator’s score aligns with other annotator scores, using Spearman’s ρ . The procedure for this was to take all possible pairs of annotators, measure their agreement using ρ , and take the average ρ across all such pairs to be the inter-annotator agreement.

6.1.4 Results

The results of the first experiment are shown in Table 6.2. As expected from the fact that **Multiply** and **Categorical** differ only in how the verb vector is learned, the results of these two models are virtually identical, outperforming both baselines and the **Additive** model

by a significant margin. However, the distance from these models to the upper bound is even greater, demonstrating that there is still a lot of progress to be made.

Model	ρ
Verb Baseline	0.08
Bigram Baseline	0.02
Add	0.04
Multiply	0.17
Categorical	0.17
UpperBound	0.40

Table 6.2: Model correlation coefficients with human judgements, first experiment. $p < 0.05$ for each ρ .

6.2 Second Experiment

This second experiment, which I initially presented in [36] with Mehrnoosh Sadrzadeh, is an extension of the first experiment to the case of sentences centred around transitive verbs, composed with a subject and an object. The results of the first experiment did not demonstrate any difference between the multiplicative model, which takes into account no syntactic information or word ordering, and the syntactically motivated categorical compositional model. By running the same experiment over a new dataset, where the relations expressed by the verb have a higher arity than in the first, I hope to demonstrate that added structure leads to better results for our syntax-sensitive model.

6.2.1 Dataset Description

The construction procedure for this dataset² is almost exactly as for the first dataset, with the following differences:

- Verbs are transitive instead of intransitive.
- For each verb pair, I select a set of subject and object nouns to use as context, as opposed to just a subject noun.

Lemmatised sentences from sample entries of this dataset are shown in Table 6.3.

²Available at <http://www.cs.ox.ac.uk/activities/compdistmeaning/GS2011data.txt>.

Sentence 1	Sentence 2
table show result	table express result
map show location	map picture location
table show result	table picture result
map show location	map express location

Table 6.3: Example entries from the transitive dataset without annotator score, second experiment.

The dataset was passed to a group of annotators, as was done for the previous dataset, who scored each pair of sentences on the same scale of 1 (not similar in meaning) to 7 (similar in meaning).

6.2.2 Evaluation Methodology

The methodology for this experiment is exactly that of the previous experiment. Models compositionally construct sentence representations, and compare them using a distance metric (all vector-based models once again used cosine similarity). The rank correlation of model scores with annotator scores is calculated using Spearman’s ρ , which is used in turn to rank models.

6.2.3 Models Compared

The models compared in this experiment are those of the first experiment, with the addition of an extra trigram-based baseline (trained with SRILM, using addition of log-probability of a sentence as a similarity metric), and the **Kronecker** variation on our categorical model, previously presented in §5.3.6. With W as the distributional semantic space for all words in the corpus, trained using the same parameters as in the first experiment, $\overrightarrow{subj}, \overrightarrow{verb}, \overrightarrow{object} \in W$ as the vectors for subject, verb and object of a sentence, respectively, and with $verb_{cat}$ as the compact representation of a transitive verb learned using the algorithm presented in the previous chapter, we have the following compositional methods:

$$\mathbf{Add} : \overrightarrow{subject\ verb\ object} = \overrightarrow{subject} + \overrightarrow{verb} + \overrightarrow{object}$$

$$\mathbf{Multiply} : \overrightarrow{subject\ verb\ object} = \overrightarrow{subject} \odot \overrightarrow{verb} \odot \overrightarrow{object}$$

$$\mathbf{Categorical} : \overrightarrow{subject\ verb\ object} = \overrightarrow{verb_{cat}} \odot (\overrightarrow{subject} \otimes \overrightarrow{object})$$

$$\mathbf{Kronecker} : \overrightarrow{subject\ verb\ object} = (\overrightarrow{verb} \otimes \overrightarrow{verb}) \odot (\overrightarrow{subject} \otimes \overrightarrow{object})$$

The upper bound **UpperBound** here is, again, the inter-annotator agreement.

6.2.4 Results

The results for the second experiment are shown in Table 6.4. The baseline scores fall in the 0.14–0.18 range, with best results being obtained for the **Trigram Baseline**. The additive model **Add** performs comparably to the additive model in the first experiment. This is most likely due to the fact that, through addition, the semantic noise caused by the overlapping word senses present in semantic vectors is increased rather than cancelled out. In other words, information from the verb vectors (namely relatively low or high values) which might be used to implicitly discriminate senses through composition is instead “drowned out” by the information from the subject and object vectors through summation, rather than being used to provide a more finely-grained representation. The multiplicative model **Multiply** performs on par with the version used for the intransitive experiment, but obtains a score comparable to the baselines.

The best performing models here are those developed and described in this thesis. The **Categorical** model now outperforms both the baselines and **Multiply**, obtaining a score of 0.21. Finally, the newly introduced **Kronecker** model leads the pack by a steady margin, with a score of 0.28.

The inter-annotator agreement **UpperBound** is much higher in this experiment than in the previous experiment, indicating even more room for improvement.

Model	ρ
Verb Baseline	0.16
Bigram Baseline	0.14
Trigram Baseline	0.18
Add	0.05
Multiply	0.17
Categorical	0.21
Kronecker	0.28
UpperBound	0.62

Table 6.4: Model correlation coefficients with human judgements, second experiment. $p < 0.05$ for each ρ .

6.3 Third Experiment

The third and final experiment I present is a modified version of the second dataset presented above, where the nouns in each entry are under the scope of adjectives applied to them. The intuition behind the datasets presented in §6.1 and §6.2 was that ambiguous verbs are disambiguated through composition with nouns. These nouns themselves may also be ambiguous, and a good compositional model will be capable of separating the noise produced by other meanings through its compositional mechanism to produce less ambiguous phrase representations. The intuition behind this dataset is similar, in that adjectives provide both additional information for disambiguation of the nouns they apply to, but also additional semantic noise. Therefore a good model will also be able to separate the useful information of the adjective from its semantic noise when composing it with its argument, in addition to doing this when composing the noun phrases with the verb.

6.3.1 Dataset Description

The construction procedure for this dataset was to take the dataset from §6.2, and, for each entry, add a pair of adjectives from those most frequently occurring in the corpus. The first adjective from the pair is applied to the first noun (subject) of the entry when forming the sentences, and the second adjective is applied to the second noun (object).

This new dataset³ was then annotated again by a group of 50 annotators using Amazon’s Mechanical Turk service, asked to give each sentence pair a meaning similarity score between 1 and 7, as for the previous datasets. As a form of quality control, I inserted “gold standard” sentences in the form of identical sentence pairs and rejected annotators that did not score these gold standard sentences with a high score of 6 or 7. Some 94 users returned annotations, of which I kept 50 according to gold standard tests. I did not apply this “gold standard” quality test to the second dataset as it was annotated by friends, and am unaware of whether or not it was applied in the production of the first dataset, but believe that this can only lead to the production of higher quality annotations.

Sample sentences from this dataset are shown in Table 6.5.

6.3.2 Evaluation Methodology

The evaluation methodology in this experiment is identical to that of the previous experiments.

³Available at <http://www.cs.ox.ac.uk/activities/compdistmeaning/GS2012data.txt>.

Sentence 1	Sentence 2
statistical table show good result	statistical table express good result
statistical table show good result	statistical table depict good result

Table 6.5: Example entries from the adjective-transitive dataset without annotator score, third experiment.

6.3.3 Models Compared

In this experiment, in lieu of simply comparing compositional models “across the board”, e.g. using the multiplicative model for both adjective-noun composition and verb-argument composition, I experimented with different *combinations of models*. This evaluation procedure was chosen because I believe that adjective-noun composition need not necessarily be the same kind of compositional process as subject-verb-object composition, but also because different models may latch onto different semantic features during the compositional process, and it would be interesting to see what model mixtures work well together.

Each mixed model has two components: a verb-argument composition model and an adjective-noun composition model. For verb-argument composition, I used the three best models from the previous experiment, namely **Multiply**, **Categorical** and **Kronecker**. For adjective-noun composition I used three different methods. With $\overrightarrow{adjective}$ and \overrightarrow{noun} being the vectors for an adjective and a noun in the distributional lexical semantic space W (built using the same procedure as the previous experiments) and $\overrightarrow{adj_{cat}}$ being the compact representation in the **Categorical** model, built according to the algorithm from §5.3, we have the following models:

$$\begin{aligned} \mathbf{AdjMult} : \overrightarrow{adjective\ noun} &= \overrightarrow{adjective} \odot \overrightarrow{noun} \\ \mathbf{Categorical} : \overrightarrow{adjective\ noun} &= \overrightarrow{adj_{cat}} \odot \overrightarrow{noun} \end{aligned}$$

The third model, **AdjNoun**, is a holistic (non-compositional) model, wherein the adjective-noun compound was treated as a single token, as its semantic vector $\overrightarrow{(adjective\ noun)_{lex}} \in W$ was learned from the corpus using the same learning procedure applied to construct other vectors in W . Hence the model defines adjective-noun “composition” as:

$$\mathbf{AdjNoun} : \overrightarrow{adjective\ noun} = \overrightarrow{(adjective\ noun)_{lex}}$$

In addition to these models, I also evaluated three baselines: **Verb Baseline**, **Bigram Baseline**, and **Trigram Baseline**. As in previous experiments, the verb baseline uses the

verb vector as a sentence vector, ignoring the information provided by other words. The bigram and trigram baselines are calculated from the same language model as used in the second experiment. In both cases, the log-probability of each sentence is calculated using SRLIM, and the sum of log-probabilities of two sentences is used as a similarity measure.

Finally, three addition-based models of sentence formation were evaluated as well, using basic linear algebraic operations. With

$$\overrightarrow{sentence} = \overrightarrow{adjective_1 \ noun_1 \ verb \ adjective_2 \ noun_2}$$

these are:

$$\mathbf{Additive} : \overrightarrow{sentence} = \overrightarrow{adjective_1} + \overrightarrow{noun_1} + \overrightarrow{verb} + \overrightarrow{adjective_2} + \overrightarrow{noun_2}$$

$$\mathbf{AddMult} : \overrightarrow{sentence} = (\overrightarrow{adjective_1} + \overrightarrow{noun_1}) \odot \overrightarrow{verb} \odot (\overrightarrow{adjective_2} + \overrightarrow{noun_2})$$

$$\mathbf{MultAdd} : \overrightarrow{sentence} = (\overrightarrow{adjective_1} \odot \overrightarrow{noun_1}) + \overrightarrow{verb} + (\overrightarrow{adjective_2} \odot \overrightarrow{noun_2})$$

6.3.4 Results

The results for the third experiment are shown in Table 6.6. Going through the combined models, we can notice that in most cases the results stay the same whether the adjective-noun combination method is **AdjMult** or **CategoricalAdj**. This is because, as was shown in the first experiment, composition of a unary-relation such as an adjective or intransitive verb with its sole argument under the categorical model with reduced representations is mathematically equivalent to the multiplicative model. The sole difference is the way the adjective or intransitive verb vector is constructed. I note, however, that with **Categorical** as a verb-argument composition method, the **CategoricalAdj** outperforms **AdjMult** by a non-negligible margin (0.19 vs. 0.14), indicating that the difference in learning procedure can lead to different results depending on what other models it is combined with.

Overall, the best results are obtained for **AdjMult+Kronecker** ($\rho = 0.26$) and **CategoricalAdj+Kronecker** ($\rho = 0.27$). Combinations of the adjective composition methods with other composition methods than the two listed above at best matches the best-performing baseline, **Verb Baseline**. In all cases, the holistic model **AdjNoun** provides the worst results.

Finally, I note that the fairly simple model **AddMult** trails the best performing models by only a few points.

Model	ρ
Verb Baseline	0.20
Bigram Baseline	0.14
Trigram Baseline	0.16
Additive	0.10
AddMult	0.24
MultAdd	0.05
Multiplicative	
AdjMult	0.20
AdjNoun	0.05
CategoricalAdj	0.20
Categorical	
AdjMult	0.14
AdjNoun	0.16
CategoricalAdj	0.19
Kronecker	
AdjMult	0.26
AdjNoun	0.17
CategoricalAdj	0.27
Upperbound	0.48

Table 6.6: Model correlation coefficients with human judgements, third experiment. $p < 0.05$ for each ρ .

6.4 Discussion

In this Chapter, I evaluated the framework described in Chapter 3, implemented using the learning procedures described in Chapter 5, and variants described earlier in this Chapter against other unsupervised compositional distributional models. I used non-compositional models and n -gram language models as baselines. These experiments show that the concrete categorical model developed here, and the Kronecker product-based variant presented alongside it, outperform all other models in each experiment save the first, where they perform on par with the leading model. As the experiments involved progressively more syntactically complicated sentences, the increased reliability of the categorical approaches relative to competing models as sentence complexity rises seems to indicate that both the Categorical and Kronecker models successfully leverage the added information provided by additional terms and syntactic structures. While these experiments demonstrate that the added structure provided by the categorical passage from syntax to semantics yields improvements in the quality of semantic representations obtained through composition, they

also reveal that there is still significant ground to cover in order to approach levels of performance comparable to that of human annotators.

The third experiment also served to show that using different combinations of composition operations depending on the syntactic type of the terms being combined can yield better results, and that some models combine better than others. Notably, the adjective-noun combination models **AdjMult** and **CategoricalAdj**, despite their mathematical similarity, produce noticeably different results when combined with the categorical verb-argument composition operation, while they perform equally with most other verb-argument composition operations. Likewise, the combination of additive and multiplicative models in **AddMult** performs surprisingly well, despite its simplicity (although the choice of operations was chosen by hand, and there is no canonical way of extending this to every sentence structure). We can conclude that different models combine different semantic aspects more prominently than others, and that through combination we can obtain better representations by assuming that different kinds of composition play on different semantic properties. For example, predicates such as intersective adjectives often add information to their argument (a red ball is a ball that is also red), hence it may be no surprise that using addition in **AddMult** performs better than across-the-board component-wise multiplication. This opens the question of how to design models of composition that systematically select which operations will match the semantic aspects of the words being combined based on their syntactic type. This is an open question, which I believe warrants further investigation.

Chapter 7

Further Work

Chapter Abstract

This chapter presents three areas in which further work has been undertaken, based on the foundations established by the rest of this thesis. It describes the issues surrounding the integration of logic into compositional distributional models, presents a new machine learning algorithm for learning semantic representations, and outlines how Combinatory Categorical Grammars may be integrated into the DisCoCat framework. It concludes by suggesting directions future work might take in this field.

Before concluding this thesis, let us examine some topics which have constituted further research on this topic, both from my own work and from work done in collaboration with colleagues. In this chapter, I will examine three areas in which further developments of the DisCoCat framework and models stemming from it have occurred, before discussing how these three areas exemplify and provide a foundation for future work. In §7.1, I will discuss the issues surrounding the modelling of logical operators within the DisCoCat framework, and what difficulties our reliance on multilinear maps modelled as tensors gives rise to. In §7.2, I will report new work done on the topic of integrating sophisticated machine learning methods into the learning procedures for our semantic representations. In §7.3, I will present the foundations for further syntactic extensions allowing us to integrate a more expressive grammatical formalism, Combinatory Categorical Grammar, into the DisCoCat framework. Finally, in §7.4, I conclude by showing how these three areas of research set the tone for future work to be done on the topic of categorical compositional distributional semantics.

7.1 Distributional Logic

The DisCoCat formalism of [19] presented in Chapter 3 and the models derived from it presented in Chapter 5 allow us to represent and learn syntactically-motivated semantic objects which can be composed to form distributional sentence representations. The learning algorithms provided allow us to model a large class of word-types, technically including logical words present in every day language use, such as “and”, “or”, and “not”. However, this closed lexical class has a very specific use in language, which historically has been modelled as operations in propositional or predicate logical calculus. As such, it may seem counter-intuitive to learn these from data; in fact, it would be more reasonable to take the finite class of words used to express quantification, conjunction, implication, and other logical concepts, as semantic objects which are *designed*.

A good question following this observation might therefore be: how can logic be modelled in a compositional distributional semantic model? Furthermore, how would logical operators work when applied to non-truth theoretic representations such as those developed in Chapter 5? This second question is an open problem I leave for future work, but to even begin to answer it, it would be helpful to see how classical predicate logic can be simulated within the DisCoCat framework (or something similar). The hope is that we could use such a simulation as a basis or inspiration for the development of non-truth theoretic distributional logic operations in future research. In this section, I discuss how such a logic might be simulated, based on work presented in [34], as well as the issues faced by our models in doing so.

7.1.1 Distributional Formal Semantics

A popular approach to compositionality in formal semantics is to derive a formal representation of a phrase from its grammatical structure and a set of associated rules. I represent the semantics of words as functions and arguments, and use the grammatical structure to dictate the order and scope of function application. For example, formal semantic models in the style of [65] will associate a semantic rule with each syntactic rule in a context-free grammar. A simple formal semantic model is shown in Figure 7.1.

Following these rules shown, the parse of a simple sentence like ‘angry dogs chase furry cats’ yields the following interpretation: $\llbracket \textit{chase} \rrbracket (\llbracket \textit{furry} \rrbracket (\llbracket \textit{cats} \rrbracket), \llbracket \textit{angry} \rrbracket (\llbracket \textit{dogs} \rrbracket))$. This model is very simplistic, and typically, a higher order logic such as a lambda calculus will be used to provide additional structure, but the key aspect retained by this simple model is that the grammar dictates the translation from natural language to the functional form. Generally, in formal semantic models, such functions will be logical relations and pred-

Syntax	Semantics	Syntax (cont'd)	Semantics (cont'd)
$S \Rightarrow NP VP$	$\llbracket S \rrbracket \Rightarrow \llbracket VP \rrbracket (\llbracket NP \rrbracket)$	$Vt \Rightarrow \{verbs_t\}$	$\llbracket Vt \rrbracket \Rightarrow \llbracket verb_t \rrbracket$
$NP \Rightarrow N$	$\llbracket NP \rrbracket \Rightarrow \llbracket N \rrbracket$	$Vi \Rightarrow \{verbs_i\}$	$\llbracket Vi \rrbracket \Rightarrow \llbracket verb_i \rrbracket$
$N \Rightarrow ADJ N$	$\llbracket N \rrbracket \Rightarrow \llbracket ADJ \rrbracket (\llbracket N \rrbracket)$	$ADJ \Rightarrow \{adjs\}$	$\llbracket ADJ \rrbracket \Rightarrow \llbracket adj \rrbracket$
$VP \Rightarrow Vt NP$	$\llbracket VP \rrbracket \Rightarrow \llbracket Vt \rrbracket (\llbracket NP \rrbracket)$	$N \Rightarrow \{nouns\}$	$\llbracket N \rrbracket \Rightarrow \llbracket noun \rrbracket$
$VP \Rightarrow Vi$	$\llbracket VP \rrbracket \Rightarrow \llbracket Vi \rrbracket$		

Figure 7.1: A simple formal semantic model.

icates applied to arguments, namely objects from the logical domain. In this section, I consider a way of defining such functions as multilinear maps over geometric objects instead. This framework is generally applicable to other formal semantic models than that which is presented here.

7.1.2 Tensors as Functions

The bijective correspondence between linear maps and matrices is a well known property in linear algebra: every linear map $f : A \rightarrow B$ can be encoded as a $dim(B)$ by $dim(A)$ matrix M , and conversely every such matrix encodes a class of linear maps determined by the dimensionality of the domain and co-domain. The application of a linear map f to a vector $\vec{v} \in A$ producing a vector $\vec{w} \in B$ is equivalent to the matrix multiplication:

$$f(\vec{v}) = M \times \vec{v} = \vec{w}$$

This correspondence generalises for multilinear maps to a correlation between n -ary maps and order $n + 1$ tensors [6, 56].

Tensors are best described as a generalisation of the notion of vectors and matrices to *larger degrees of freedom* referred to as tensor orders (one for vectors, two for matrices). To illustrate this generalisation, consider how vectors, which are order 1 tensors, may be written as the weighted superposition (summation) of their basis elements: for some vector space V with basis $\{\vec{b}_i\}_i$, any vector $\vec{v} \in V$ can be written

$$\vec{v} = \sum_i c_i^v \vec{b}_i = [c_1^v, \dots, c_i^v, \dots, c_{dim(V)}^v]$$

where the weights c_i^v are elements of the underlying field (e.g. \mathbb{R}), and thus vectors can be fully described by such a one-index summation. Likewise, matrices, which are order 2 tensors, can be seen as a collection of row vectors from some space V_r with basis $\{\vec{a}_i\}_i$, or of column vectors from some space V_c with basis $\{\vec{d}_j\}_j$. Such a matrix M is an element of

the space $V_r \otimes V_c$, and can be fully described by the two index summation:

$$M = \sum_{ij} c_{ij}^M \vec{a}_i \otimes \vec{d}_j$$

where, once again, c_{ij}^M is an element of the underlying field which in this case is simply the element from the i th row and j th column of the matrix M , and the basis element $\vec{a}_i \otimes \vec{d}_j$ of $V_r \otimes V_c$ is formed by a pair of basis elements from V_r and V_c . The number of indices (or degrees of freedom) used to fully describe a tensor in this superposition notation is its order, e.g., an order 3 tensor $T \in A \otimes B \otimes C$ would be described by the superposition of weights c_{ijk}^T associated with basis elements $\vec{e}_i \otimes \vec{f}_j \otimes \vec{g}_k$.

Matrix multiplications, inner products, and traces all generalise to tensors as the non-commutative tensor contraction operation (\times). For tensors $T \in A \otimes \dots \otimes B \otimes C$ and $U \in C \otimes D \otimes \dots \otimes E$, with bases $\{\vec{a}_i \otimes \dots \otimes \vec{b}_j \otimes \vec{c}_k\}_{i\dots jk}$ and $\{\vec{c}_k \otimes \vec{d}_l \otimes \dots \otimes \vec{e}_m\}_{kl\dots m}$, the tensor contraction of $T \times U$ is calculated:

$$\sum_{i\dots jkl\dots m} c_{i\dots jk}^T c_{kl\dots m}^U \vec{a}_i \otimes \dots \otimes \vec{b}_j \otimes \vec{d}_l \otimes \dots \otimes \vec{e}_m$$

where the resulting tensor is of order equal to two less than the sum of the orders of the input tensors; the subtraction reflects the elimination of matching basis elements through summation during contraction.

For every curried multilinear map $g : A \rightarrow \dots \rightarrow Y \rightarrow Z$, there is a tensor $T^g \in Z \otimes Y \otimes \dots \otimes A$ encoding it [6, 56]. The application of a curried n -ary map $h : V_1 \rightarrow \dots \rightarrow V_n \rightarrow W$ to input vectors $\vec{v}_1 \in V_1, \dots, \vec{v}_n \in V_n$ to produce output vector $\vec{w} \in W$ corresponds to the tensor contraction of the tensor $T^h \in W \otimes V_n \otimes \dots \otimes V_1$ with the argument vectors:

$$h(\vec{v}_1) \dots (\vec{v}_n) = T^h \times \vec{v}_1 \times \dots \times \vec{v}_n$$

This isomorphism between tensors (objects) and multilinear maps (maps) is also known in the quantum information theory literature (e.g. [91, 10]) as *map-state duality*, whereby for each map there is a bi-partite entangled state, with the map and state each forming part of the dual representation of the same piece of information.

7.1.3 Formal Semantics with Tensors

Using the correspondence between n -ary maps and tensors of order $n+1$ discussed in §7.1.2, and the correspondence between function arguments and semantic vectors in standard dis-

tributional semantic models, we can turn any formal semantic model into a compositional distributional semantic model. This is done by first running a type inference algorithm on the semantic interpretations of generative rules and obtaining both basic (i.e. uninferred) and function (i.e. inferred) types, then assigning to each basic type a vector space and to each function type a tensor space. Following this, we can represent arguments by vectors and functions by tensors, and finally, we can model function application by tensor contraction.

To give an example, in the simple formal semantic model presented in Figure 7.1, a type inference algorithm in the vein of [48, 61] would give the following type assignments:

- It would leave the types of $\llbracket N \rrbracket$ and $\llbracket S \rrbracket$ uninferred, as they are not treated as functions in any rule; therefore, we treat nouns and sentences as vectors in some spaces N and S .
- Noun phrases are simply assigned the same type as $\llbracket N \rrbracket$, hence elements of $\llbracket NP \rrbracket$ would also be vectors in N .
- Verb phrases map noun phrase interpretations to sentence interpretations, hence $\llbracket VP \rrbracket : type(\llbracket NP \rrbracket) \rightarrow type(\llbracket S \rrbracket)$ or, as we have inferred the relevant spaces, $\llbracket VP \rrbracket : N \rightarrow S$.
- Intransitive verb interpretations are assigned the same types as verb phrase interpretations, and hence an intransitive verb “vi” can be represented by some tensor $T^{vi} \in S \otimes N$.
- Transitive verbs are interpreted as maps $\llbracket Vt \rrbracket : type(\llbracket NP \rrbracket) \rightarrow type(\llbracket VP \rrbracket)$, which can be expanded to $\llbracket Vt \rrbracket : N \rightarrow N \rightarrow S$, giving us, for some transitive verb “vt”, the tensor form $T^{vt} \in S \otimes N \otimes N$.
- Finally, adjectives are maps of form $\llbracket Adj \rrbracket : type(\llbracket N \rrbracket) \rightarrow type(\llbracket N \rrbracket)$, and hence some adjective “adj” has the tensor form $T^{adj} \in N \otimes N$.

Putting all this together with tensor contraction (\times) as function application, let us examine the computation of a sample sentence “angry dogs chase furry cats” with interpretation $\llbracket chase \rrbracket (\llbracket furry \rrbracket (\llbracket cats \rrbracket), \llbracket angry \rrbracket (\llbracket dogs \rrbracket))$. This would simply be the calculation of $(T^{chase} \times (T^{furry} \times T^{cats})) \times (T^{angry} \times T^{dogs})$, where T^{cats} and T^{dogs} would be lexical semantic vectors, T^{furry} and T^{angry} would be square matrices, and T^{chase} would be an order 3 tensor.

It should be noted that the semantic tensors generated in this framework are in fact identical to those found in the DisCoCat framework, as tensor contraction is just iterated

application of ϵ maps. To give an example, let T^f be tensor in $C \otimes B \otimes A$ modelling a multilinear map $f : A \times B \rightarrow C$. Let $\vec{v} \in A$ and $\vec{w} \in B$ be its arguments. Then the tensor contraction corresponding to the application of T^f to its arguments can be expressed as ϵ maps as follows:

$$(T^f \times \vec{v}) \times \vec{w} = ((1_C \otimes \epsilon_B) \circ (1_C \otimes \epsilon_A \otimes 1_B))(T^f \otimes \vec{v} \otimes \vec{w})$$

The principal (superficial) difference between this framework and DisCoCat is that tensors always ‘absorb’ arguments on the right through contraction, forming a linear λ -calculus, whereas in DisCoCat, contraction can happen on the left of a tensor. This difference is indeed superficial, in that any tensor can be transposed to ‘take arguments’ on the left or the right, thereby allowing us to produce an isomorphism between the tensors in a DisCoCat model and those in this distributional formal semantics framework, provided that the vector spaces match.

7.1.4 Simulating Simple Predicate Calculi

The approach described above allows us to adapt very simple formal semantic models to work with tensors in a way similar to the DisCoCat framework discussed throughout this thesis. Before turning to the question of how logical connectives and quantifiers might be modelled using tensors in either the distributional formal semantics described above, or in the DisCoCat framework, I begin by showing how tensors can represent model-theoretic predicates from a finite-domain predicate calculus.

In [19], a short example shows how simple predicate logic can be simulated within the DisCoCat framework by setting S to a boolean space. This can be done in one of two ways: namely either by setting S to be a one dimensional space B_1 with basis vector $\vec{\top}$, where the vector $\vec{\top} \equiv \top$ and $\vec{0} \equiv \perp$; or by setting S to be a two dimensional space B_2 with basis $\{\vec{\top}, \vec{\perp}\}$, where:

$$\vec{\top} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \equiv \top \quad \vec{\perp} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \equiv \perp$$

To represent a logical model in vector spaces using either of the above options for boolean space, I consider the following translation mechanism:

- I assign to the domain \mathcal{D} —the set of objects in our logic—a vector space N over \mathbb{N} of $|\mathcal{D}|$ dimensions where each basis vector of N is in one-to-one correspondence with elements of \mathcal{D} .
- An element of \mathcal{D} is therefore represented as a one-hot vector in N (a sparse unit-

length vector with a single non-zero value), the single-non null value of which is the weight for the basis vector mapped to that element of \mathcal{D} .

- Similarly, a subset of \mathcal{D} is a vector of N where those elements of \mathcal{D} in the subset have 1 as their corresponding basis weights in the vector, and those not in the subset have 0. Therefore there is a one-to-one correspondence between the vectors in N with basis weights of either 0 or 1 and the elements of the power set $\mathcal{P}(\mathcal{D})$.
- Each unary predicate P in the logic is represented in the logical model as a set $M_P \subseteq \mathcal{D}$ containing the elements of the domain for which the predicate is true. Traditionally, we could view the predicate as a function $f_P : \mathcal{D} \rightarrow \mathbb{B}$ where:

$$f_P(x) = \begin{cases} \top & \text{if } x \in M_P \\ \perp & \text{otherwise} \end{cases}$$

Given that our sentence space is set to a boolean space, using one of the forms suggested above (i.e. B_1 or B_2), we could directly model this function as a tensor in $S \otimes N$ as discussed in §7.1.3. However, I suggest a different way of distributionally modelling predicates which is more in line with how predicates work in a DisCoCat. Let us instead view predicates P as functions $f_P : \mathcal{P}(\mathcal{D}) \rightarrow \mathbb{B}$, defined as:

$$f_P(X) = X \cap M_P$$

Therefore the distributional form of these functions will be tensors in $N \otimes N$. Through tensor contractions, they map subsets of \mathcal{D} (elements of N) to subsets of \mathcal{D} containing only those objects of the original subset for which P holds (i.e. yielding another vector in N).

- Finally, n -ary relations R such as verbs are represented in a logical model as the set M_R of n -tuples of elements from \mathcal{D} for which R holds. Therefore such relations can be represented as functions $f_R : \mathcal{D}^n \rightarrow \mathbb{B}$ where:

$$f_R(x_1, \dots, x_n) = \begin{cases} \top & \text{if } (x_1, \dots, x_n) \in M_R \\ \perp & \text{otherwise} \end{cases}$$

Considering our choice of S as a boolean space, and using the process described in §7.1.3, we can represent these relations distributionally as tensors in $S \otimes \underbrace{N \otimes \dots \otimes N}_n$.

To give a concrete example about how predicates and relations (namely verbs) would work in our setting, let our domain be the individuals John (j), Mary (m) and Peter (p). John

and Mary love each other and love themselves, but are indifferent about Peter. Peter hates himself and John, but loves Mary. The logical model for this budding romantic tragedy is as follows:

$$\begin{aligned}\mathcal{D} &= \{j, m, p\} \\ M_{\text{loves}} &= \{(j, j), (m, m), (j, m), (m, j), (p, m)\} \\ M_{\text{hates}} &= \{(p, p), (p, j)\}\end{aligned}$$

Distributionally speaking, the elements of the domain will be mapped to the following one-hot vectors in some three-dimensional space N as follows:

$$j \mapsto \vec{j} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad m \mapsto \vec{m} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad p \mapsto \vec{p} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The representation of the verbs will depend on our choice of sentence space S . If we set $S = B_1$, we define the verbs as follows, using the Kronecker product:

$$\begin{aligned}f_{\text{loves}} &\mapsto (\vec{1} \otimes \vec{j} \otimes \vec{j}) + (\vec{1} \otimes \vec{m} \otimes \vec{m}) + (\vec{1} \otimes \vec{j} \otimes \vec{m}) + (\vec{1} \otimes \vec{m} \otimes \vec{j}) + (\vec{1} \otimes \vec{p} \otimes \vec{m}) \\ f_{\text{hates}} &\mapsto (\vec{1} \otimes \vec{p} \otimes \vec{p}) + (\vec{1} \otimes \vec{p} \otimes \vec{j})\end{aligned}$$

Using the distributivity of \otimes over $+$ we can express this more compactly as:

$$\begin{aligned}f_{\text{loves}} &\mapsto T^{\text{loves}} = \vec{1} \otimes ((\vec{j} \otimes \vec{j}) + (\vec{m} \otimes \vec{m}) + (\vec{j} \otimes \vec{m}) + (\vec{m} \otimes \vec{j}) + (\vec{p} \otimes \vec{m})) \\ f_{\text{hates}} &\mapsto T^{\text{hates}} = \vec{1} \otimes ((\vec{p} \otimes \vec{p}) + (\vec{p} \otimes \vec{j}))\end{aligned}$$

These definitions implicitly set the basis weight for sets of elements not in M_{loves} and M_{hates} to zero.

If we set $S = B_2$ we simply have to explicitly state all the element pairs for which the relation is false, in addition to those pairs for which it is true. Using the more compact notation used above, this gives:

$$\begin{aligned}f_{\text{loves}} &\mapsto T^{\text{loves}} = \vec{1} \otimes ((\vec{j} \otimes \vec{j}) + (\vec{m} \otimes \vec{m}) + (\vec{j} \otimes \vec{m}) + (\vec{m} \otimes \vec{j}) + (\vec{p} \otimes \vec{m})) \\ &\quad + \vec{1} \otimes ((\vec{p} \otimes \vec{p}) + (\vec{m} \otimes \vec{p}) + (\vec{j} \otimes \vec{p}) + (\vec{p} \otimes \vec{j})) \\ f_{\text{hates}} &\mapsto T^{\text{hates}} = \vec{1} \otimes ((\vec{p} \otimes \vec{p}) + (\vec{p} \otimes \vec{j})) \\ &\quad + \vec{1} \otimes ((\vec{j} \otimes \vec{j}) + (\vec{m} \otimes \vec{m}) + (\vec{j} \otimes \vec{m}) + (\vec{m} \otimes \vec{j}) + (\vec{j} \otimes \vec{p}) + (\vec{m} \otimes \vec{p}) + (\vec{p} \otimes \vec{m}))\end{aligned}$$

Computing the value of a sentence such as “John loves Mary” would then involve producing the distributional representation of $f_{\text{loves}}(j, m)$. In the distributional setting, this would be equivalent to the following computation with $S = B_1$, and with \times as the tensor contraction operation:

$$\begin{aligned}
& (T^{\text{loves}} \times \vec{m}) \times \vec{j} \\
&= ((\vec{\tau} \otimes ((\vec{m} \otimes \vec{m}) + (\vec{j} \otimes \vec{m}) + (\vec{p} \otimes \vec{m}))) \times \vec{m}) \times \vec{j} \\
&= (\vec{\tau} \otimes (\vec{m} + \vec{j} + \vec{p})) \times \vec{j} \\
&= (\vec{\tau} \otimes \vec{j}) \times \vec{j} \\
&= \vec{\tau}
\end{aligned}$$

In contrast, “Peter hates Mary” would yield the computation:

$$\begin{aligned}
& (T^{\text{hates}} \times \vec{m}) \times \vec{p} \\
&= ((\vec{\tau} \otimes ((\vec{p} \otimes \vec{p}) + (\vec{p} \otimes \vec{j}))) \times \vec{m}) \times \vec{p} \\
&= \vec{\tau} \cdot 0 \times \vec{p} \\
&= \vec{\tau} \cdot 0 \\
&= \vec{0}
\end{aligned}$$

which is the origin in $S = B_1$, which I interpret as false. In the case of false statements, the nature of the computation is a lot less clear mathematically. In contrast, it is explicit in the case of $S = B_2$, for which we reproduce both computations initially shown above here. For “John loves Mary” we can notice very little change, since everybody loves Mary:

$$\begin{aligned}
& (T^{\text{loves}} \times \vec{m}) \times \vec{j} \\
&= ((\vec{\tau} \otimes ((\vec{m} \otimes \vec{m}) + (\vec{j} \otimes \vec{m}) + (\vec{p} \otimes \vec{m})) + \vec{\perp} \cdot 0 \times \vec{m}) \times \vec{j} \\
&= (\vec{\tau} \otimes (\vec{m} + \vec{j} + \vec{p}) + \vec{\perp} \cdot 0) \times \vec{j} \\
&= (\vec{\tau} \otimes \vec{j} + \vec{\perp} \cdot 0) \times \vec{j} \\
&= \vec{\tau} \cdot 1 + \vec{\perp} \cdot 0 \\
&= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\
&= \vec{\tau}
\end{aligned}$$

But for “Peter hates Mary” the falsehood of the statement is more clearly computed:

$$\begin{aligned}
& (T^{\text{hates}} \times \vec{m}) \times \vec{p} \\
&= ((\vec{i} \otimes ((\vec{p} \otimes \vec{p}) + (\vec{p} \otimes \vec{j}))) + \vec{1} \otimes ((\vec{m} \otimes \vec{m}) + (\vec{j} \otimes \vec{m}) + (\vec{p} \otimes \vec{m}))) \times \vec{m} \times \vec{p} \\
&= (\vec{i} \cdot 0 + \vec{1} \otimes (\vec{m} + \vec{j} + \vec{p})) \times \vec{p} \\
&= \vec{i} \cdot 0 + \vec{1} \cdot 1 \\
&= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\
&= \vec{1}
\end{aligned}$$

Let us quickly illustrate how predicates work distributionally: let us consider a domain with two dogs (a and b) and a cat (c). One of the dogs is brown, as is the cat. Let D be the set of dogs, and B the predicate “brown”. I represent these statements in the model as follows:

$$\mathcal{D} = \{a, b, c\}$$

$$D = \{a, b\}$$

$$M_B = \{b, c\}$$

Distributionally, I model the domain as a three dimensional vector space, the set of dogs as a vector

$$D \mapsto \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

and the predicate ‘brown’ as a tensor in $N \otimes N$

$$f_B \mapsto T^{\text{brown}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The set of brown dogs is obtained by computing $f_B(D)$, which distributionally corresponds to applying the tensor T^{brown} to the vector representation of D via tensor contraction, as follows:

$$T^{\text{brown}} \times D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \vec{b}$$

Therefore using tensors and vectors, we can effectively simulate a simple predicate calculus within either the DisCoCat framework or the formal distributional semantics presented earlier in this chapter.

7.1.5 Logical Operations, and Integrating Non-Linearity

In §7.1.3, I presented a way of combining tensor-based compositional distributional models with simple formal semantic models. In §7.1.4, I showed how a simple predicate logic could be simulated within such a distributional formal semantics or the DisCoCat framework discussed throughout this thesis. However, both in the case of formal semantics and in the natural language statements we wish to model the semantics of with DisCoCat models, there are not only predicates and relations, but also logical words such ‘and’, ‘or’ and expressions such as ‘every’, ‘some’, ‘for all’, which we model logically as logical connectives (\wedge , \vee , \neg , and so on) and as quantifiers (\forall , \exists). How are these to be represented within tensor-based models? In this subsection, I show that the linear maps encoded by tensors do not *prima-facie* suffice to simulate all these logical elements, but that the distributional formal semantics described above can be enhanced with non-linear operations that fit the bill.

Logical Operations Consider the following additional rules for the formal semantics presented in Figure 7.1:

Syntax	Semantics
$S \Rightarrow S_1 \text{ and } S_2$	$\llbracket S \rrbracket \Rightarrow \wedge (\llbracket S_1 \rrbracket, \llbracket S_2 \rrbracket)$
$S \Rightarrow S_1 \text{ or } S_2$	$\llbracket S \rrbracket \Rightarrow \vee (\llbracket S_1 \rrbracket, \llbracket S_2 \rrbracket)$
$S \Rightarrow \text{not } S$	$\llbracket S \rrbracket \Rightarrow \neg (\llbracket S \rrbracket)$

Can these additional semantic functions, corresponding to conjunction, disjunction and negation be modelled by linear maps (and thereby represented as tensors)?

I considered two options for a boolean sentence space, B_1 and B_2 , as suggested by [19]. For B_1 it is trivial to show that negation cannot be modelled by a linear map. Assume there is some linear map $not : B_1 \rightarrow B_1$ such that:

$$\begin{aligned} not(\vec{1}) &= \vec{0} \\ not(\vec{0}) &= \vec{1} \end{aligned}$$

All linear maps f must satisfy

$$f(\vec{0}) = \vec{0}$$

so *not* is not a linear map, and hence no tensor models negation for $S = B_1$. Similarly, the obvious representations for conjunction and disjunction, namely

$$\text{and}(\vec{v}, \vec{w}) = \min(\vec{v}, \vec{w}) \qquad \text{or}(\vec{v}, \vec{w}) = \max(\vec{v}, \vec{w})$$

where *min* and *max* are component-wise minimum and maximum operators, are not multi-linear maps.

For $S = B_2$, [19] show that the swap matrix

$$T^{\text{not}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

models negation, which can easily be verified:

$$\begin{aligned} T^{\text{not}} \times \vec{\top} &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \vec{\perp} \\ T^{\text{not}} \times \vec{\perp} &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \vec{\top} \end{aligned}$$

I go further here and show that various other logical operators can be modelled with $S = B_2$. To make talking about order-3 tensors used to model binary operations easier, I will use the following block matrix notation for $2 \times 2 \times 2$ order-3 tensors T :

$$T = \left[\begin{array}{cc|cc} a_1 & b_1 & a_2 & b_2 \\ c_1 & d_1 & c_2 & d_2 \end{array} \right]$$

which allows us to express tensor contractions as follows, for some $\vec{v} = [\alpha \beta]^T$:

$$T \times \vec{v} = \left[\begin{array}{cc|cc} a_1 & b_1 & a_2 & b_2 \\ c_1 & d_1 & c_2 & d_2 \end{array} \right] \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha \cdot a_1 + \beta \cdot a_2 & \alpha \cdot b_1 + \beta \cdot b_2 \\ \alpha \cdot c_1 + \beta \cdot c_2 & \alpha \cdot d_1 + \beta \cdot d_2 \end{bmatrix}$$

or more concretely:

$$\begin{aligned} T \times \vec{\top} &= \left[\begin{array}{cc|cc} a_1 & b_1 & a_2 & b_2 \\ c_1 & d_1 & c_2 & d_2 \end{array} \right] \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix} \\ T \times \vec{\perp} &= \left[\begin{array}{cc|cc} a_1 & b_1 & a_2 & b_2 \\ c_1 & d_1 & c_2 & d_2 \end{array} \right] \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix} \end{aligned}$$

Using this notation, we can define tensors for the following operations:

$$\begin{aligned}
 (\neg) \mapsto T^\neg &= \left[\begin{array}{cc|cc} 0 & 1 & & \\ 1 & 0 & & \end{array} \right] \\
 (\vee) \mapsto T^\vee &= \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] \\
 (\wedge) \mapsto T^\wedge &= \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{array} \right] \\
 (\rightarrow) \mapsto T^\rightarrow &= \left[\begin{array}{cc|cc} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{array} \right]
 \end{aligned}$$

The design behind these tensors is not particularly complicated. They are applied to two arguments: if the first argument is true ($\vec{\top}$) then the left matrix of the block matrix is applied to the second argument; if the first argument is false ($\vec{\perp}$) then the right matrix of the block matrix is applied to the second argument. It is fairly trivial, using the truth tables for any logical connective, to design such partitioned matrices for the tensor representation of any logical operator in a propositional calculus. For example, for the case of \wedge , the truth table states that $a \wedge b$ is true if and only if a is true and b is true. Therefore if a is true, $a \wedge b$ holds the truth value of b , and if a is false, $a \wedge b$ is false regardless of the value of b . From this, we design the tensor for \wedge as follows

$$T^\wedge = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{array} \right]$$

When a is true, the partial application of the tensor to the first argument should yield an identity matrix, as its application to the second argument should yield the truth value of b :

$$T^\wedge \times \vec{\top} = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{array} \right] \left[\begin{array}{c} 1 \\ 0 \end{array} \right] = \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right]$$

When a is false, the partial application of the tensor to the first argument should yield a matrix which ignores the truth value of the second argument, mapping both true and false to false:

$$T^\wedge \times \vec{\perp} = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{array} \right] \left[\begin{array}{c} 0 \\ 1 \end{array} \right] = \left[\begin{array}{cc} 0 & 0 \\ 1 & 1 \end{array} \right]$$

Having established these partial applications, we can verify that the truth table for \wedge is replicated:

$$\begin{aligned} (T^\wedge \times \vec{T}) \times \vec{T} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \vec{T} \\ (T^\wedge \times \vec{T}) \times \vec{\perp} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \vec{\perp} \\ (T^\wedge \times \vec{\perp}) \times \vec{T} &= \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \vec{\perp} \\ (T^\wedge \times \vec{\perp}) \times \vec{\perp} &= \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \vec{\perp} \end{aligned}$$

A similar set of steps can be used to verify the other tensor encodings of logical connectives specified above.

Non-linearity and Quantification It may therefore seem that for the case where $S = B_2$, logical operators may be dealt with using only multi-linear maps represented as tensors. However, when it comes to dealing with quantification, it is not obvious that a solution using multilinear maps exists.

An intuitive way of modelling universal quantification is as follows: expressions of the form “All X s are Y s” are true if and only if $M_X = M_X \cap M_Y$, where M_X and M_Y are the sets of X s and the set of Y s, respectively. We saw earlier that sets of objects of the domain could be represented as vectors \vec{X} and \vec{Y} in a vector space N . The intersection of two such sets can be represented distributionally using the component-wise minimum function *min*:

$$M_X \cap M_Y \mapsto \min(\vec{X}, \vec{Y})$$

Using this, we can define the map *forall* for distributional universal quantification modelling expressions of the form “All X s are Y s” as follows:

$$\text{forall}(\vec{X}, \vec{Y}) = \begin{cases} \vec{T} & \text{if } \vec{X} = \min(\vec{X}, \vec{Y}) \\ \vec{\perp} & \text{otherwise} \end{cases}$$

Existential statements of the form “There exists X ” can be modelled using the function

exists, which tests whether or not M_X is empty, and is defined as follows:

$$exists(\vec{X}) = \begin{cases} \vec{\top} & \text{if } |\vec{X}| > 0 \\ \vec{\perp} & \text{otherwise} \end{cases}$$

In both cases, the definitions are given for $S = B_2$, but are adaptable to $S = B_1$ by setting $\vec{\perp} = \vec{0}$.

To give a simple example, let us take the domain $\mathcal{D} = \{dog_1, dog_2, cat_1\}$, where dog_1 is a black dog, dog_2 is a brown dog, and cat_1 is a brown cat. Let the set \vec{B} of brown things, the set \vec{C} of cats and \vec{P} of dogs be represented in vector form as follows:

$$\vec{B} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad \vec{C} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \vec{P} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

To evaluate a sentence such as “all brown things are dogs” we would compute $forall(\vec{B}, \vec{P})$ by first computing

$$min(\vec{B}, \vec{P}) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

then checking if the result is equal to \vec{B} , which it is not, and hence $forall(\vec{B}, \vec{P}) = \vec{\perp}$. We can check if there exist any brown cats by first computing the intersection of the set of brown things and the set of cats:

$$\vec{B} \cap \vec{C} = min(\vec{B}, \vec{C}) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

We then check the size of this vector, which is 1. The definition of *exists* tells us that $exists(\vec{B} \cap \vec{C}) = \vec{\top}$.

Neither of the quantification functions defined above are multi-linear, since a multilinear function must be linear in all arguments. A counter example for *forall* is to consider the case where M_X and M_Y are empty, and multiply their vector representations by non-zero

scalar weights α and β .

$$\begin{aligned}\alpha\vec{X} &= \vec{X} \\ \beta\vec{Y} &= \vec{Y} \\ \text{forall}(\alpha\vec{X}, \beta\vec{Y}) &= \text{forall}(\vec{X}, \vec{Y}) = \vec{\top} \\ \text{forall}(\alpha\vec{X}, \beta\vec{Y}) &\neq \alpha\beta\vec{\top}\end{aligned}$$

I observe that the equations above demonstrate that *forall* is not a multilinear map. This proof holds for $S = B_1$ and $S = B_2$.

The proof that *exists* is not a multilinear map is equally trivial. Assume M_X is an empty set and α is a non-zero scalar weight:

$$\begin{aligned}\alpha\vec{X} &= \vec{X} \\ \text{exists}(\alpha\vec{X}) &= \text{exists}(\vec{X}) = \vec{\perp} \\ \text{exists}(\alpha\vec{X}) &\neq \alpha\vec{\perp}\end{aligned}$$

This proof holds for $S = B_2$.

To conclude this section, we have seen how the two choices for boolean sentence space differ when we consider how to ‘implement’ logical operations as tensors. For $S = B_1$, most operations need be non-linear maps, which can be defined in our distributional formal semantic models but not in those constructed within the DisCoCat formalism. For $S = B_2$, logical connectives can be modelled as multilinear maps (and therefore as tensors), and can thus be used in the DisCoCat setting as well as in the distributional formal semantics model presented in this section, however the (perhaps naïve) functions used to model basic quantification were shown to be non-linear maps for both settings. The question of how to approach quantification and logic in the DisCoCat framework, either using only multilinear maps or by using some semantic representation other than **FVect**, as well as the question of how to go from simulating predicate logic with tensors to defining (or learning) logical connectives for the non-truth theoretic compositional distributional models described in this thesis (i.e. going from vector encodings of classical logic to logic for continuous vector space models), are both difficult and fundamental questions which would merit attention in future research on this topic.

7.2 Learning Tensors by Multi-Step Linear Regression

In Chapter 5, I presented learning procedures that could be used to produce distributional representations of words and relations within the DisCoCat framework based on information provided by a corpus. A parallel effort to describe distributional compositionality has been presented in the work of Marco Baroni, Roberto Zamparelli and colleagues, which I briefly described in §2.4.2.

To quickly repeat how their approach, first presented in [5], works: nouns are vectors in some vector space N , and adjectives are square matrices in $N \otimes N$. Adjective-noun composition is simply matrix-vector multiplication. In sum, this is exactly how things work in the DisCoCat models I discussed in Chapter 5. The novel aspect of their approach is the learning procedure for adjective matrices. To learn the matrix for an adjective adj , the following steps are taken:

1. For each instance of the adjective applied to some noun n , the vector $\overrightarrow{adj\ n}$ is learned using the same procedure used to learn noun vectors.
2. Let I be the set of noun vectors adj takes as ‘input’ and O be the set of adjective-noun vectors learned in the previous steps. Therefore for each input vector in I there is an output vector in O .
3. The matrix for adj is learned through linear regression, such that the output set O' produced by multiplying the matrix for adj with the elements of I is minimally different from O .

This is an interesting machine-learning approach, with a high degree of parametric freedom: we are free to select the procedure for building the noun and adjective-noun vectors, the linear-regression algorithm used, the distance metric used to compare O and O' , what sort of dimensionality reduction techniques are applied, and so on.

It should be noted that this approach can directly be applied to how we dealt with intransitive verbs in Chapter 5, since they are also matrices in $N \otimes N$. The question is now: can this learning procedure be scaled to deal with the higher order tensors used for other constructs, such as transitive verbs? In [35]¹, in collaboration with Baroni and his colleagues, Mehrnoosh Sadrzadeh and I have produced a generalised multi-step linear regression learning procedure which permits the construction of higher order semantic tensors for use in

¹I am indebted to my collaborators Georgiana Dinu, Yao-Zhong Zhang, Mehrnoosh Sadrzadeh and Marco Baroni for their contribution to this work, and for allowing me to reproduce the contents. The figure in this section was produced by Marco Baroni. The parameters for the learning procedure were described by Georgiana Dinu.

DisCoCat models. In this section, I will first present this procedure before discussing how well tensors produced using this new learning mechanism performed in the experiments presented in Chapter 6.

7.2.1 Multi-Step Linear Regression

Multi-step regression learning is a generalisation of linear regression learning for tensors of order 3 or higher, as procedures already exist for tensors of order 1 (lexical semantic vectors) and order 2 (cf. [5]). For order 1 tensors, we suggest learning vectors using any standard lexical semantic vector learning model, and present sample parameters in §7.2.2 below. Learning order 2 tensors (matrices) can be treated as a multivariate multiple regression problem, where the matrix components are chosen to optimise (in a least squares error sense) the mapping from training instances of input (argument) and output (composed expression) vectors. Consider for example the task of estimating the components of the matrix representing an intransitive verb, that maps subject vectors to (subject-verb) sentence vectors (Baroni and Zamparelli discuss the analogous adjective-noun composition case):

$$\vec{s} = V \times \overrightarrow{subj}$$

The weights of the matrix are estimated by least-squares regression from example pairs of input subject and output sentence vectors directly extracted from the corpus. For example, the matrix for *sing* is estimated from corpus-extracted vectors representing pairs such as (*mom, mom sings*), (*child, child sings*), etc. Note that if the input and output vectors are n dimensional, we must estimate an $n \times n$ matrix, each row corresponding to a separate regression problem (the i -th row vector of the estimated matrix will provide the weights to linearly combine the input vector components to predict the i -th output vector component). Regression is a supervised technique requiring training data. However, as we are extracting the training data automatically from the corpus, this approach does not incur an extra knowledge cost with respect to unsupervised methods.

Learning tensors of higher order by linear regression involves iterative application of the linear regression learning method described above. The idea is to progressively learn the functions of arity two or higher encoded by such tensors by recursively learning the partial application of these functions, thereby reducing the problem to the same matrix-learning problem as addressed by Baroni and Zamparelli. To start with an example: the matrix-by-vector operation of [5] is a special case of the general tensor-based function application model we are proposing, where a ‘mono-argumental’ function (intransitive verbs) corresponds to a order 2 tensor (a matrix). The approach is naturally extended to bi-argumental

functions, such as transitive verbs, where the verb will be a order 3 tensor to be multiplied first by the object vector and then by the subject, to return a sentence-representing vector:

$$\vec{s} = V \times \overrightarrow{obj} \times \overrightarrow{subj}$$

The first multiplication of a $n \times n \times n$ tensor by a n -dimensional vector will return a n -by- n matrix (equivalent to an intransitive verb, as it should be: both *sings* and *eats meat* are VPs requiring a subject to be saturated). Note that given n -dimensional input vectors, the ij -th n -dimensional vector in the estimated tensor provides the weights to linearly combine the input object vector components to predict the ij -th output component of the unsaturated verb-object matrix. The matrix is then multiplied by the subject vector to obtain a n -dimensional vector representing the sentence. Again, we estimate the tensor components by linear regression on input-output examples. In the first stage, we apply linear regression to obtain examples of semi-saturated matrices representing *verb-object* constructions with a specific verb. These matrices are estimated, like in the intransitive case, from corpus-extracted examples of (subject, subject-verb-object) pairs. After estimating a suitable number of such matrices for a variety of objects of the same verb, we use pairs of corpus-derived object vectors and the corresponding estimated verb-object matrices as input-output pairs for another regression, where we estimate the verb tensor components. The estimation procedure is schematically illustrated for *eat* in Fig. 7.2. We first estimate matrices for the VPs *eat-meat*, *eat-pie* etc. by linear regression on input subject and output sentence vector pairs. We then estimate the tensor for *eat* by linear regression with the matrices estimated in the previous step as output examples, and the vectors for the corresponding objects as input examples.

We can generalise this learning procedure to functions of arbitrary arity. Consider an n -ary function $f : X_1 \rightarrow \dots \rightarrow X_n \rightarrow Y$. Let L_i be the set of i -tuples $\{w_1^j, \dots, w_i^j\}_{j \in [1, k]}$, where $k = |L_i|$, corresponding to the words which saturate the first i arguments of f in a corpus. For each tuple in some set L_i , let $f w_1^j \dots w_i^j = f_i^j : X_{i+1} \rightarrow \dots \rightarrow X_n \rightarrow Y$. Trivially, there is only one such f_0^j —namely f itself—since $L_0 = \emptyset$ (as there are no arguments of f to saturate for $i = 0$). The idea behind multi-step regression is to learn, at each step, the tensors for functions f_i^j by linear regression over the set of pairs (w_{i+1}^j, f_{i+1}^j) , where the tensors f_{i+1}^j are the expected outcomes of applying f_i^j to w_{i+1}^j and are learned during the previous step. We bootstrap this algorithm by learning the vectors in Y of the set $\{f_n^j\}_j$ by treating the word which each f_n^j models combined with the words of its associated tuple in L_n as a single token. We then learn the vector for this token from the corpus using our preferred distributional semantics method. By recursively learning the sets of functions

STEP 1: ESTIMATE VP MATRICES



STEP 2: ESTIMATE V TENSOR

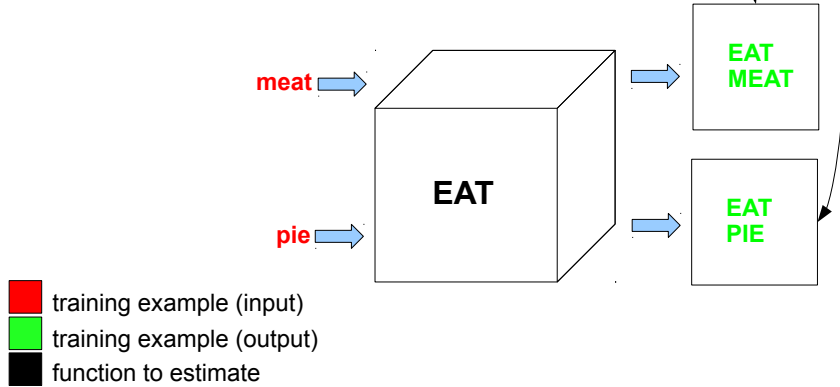


Figure 7.2: Estimating a tensor for *eat* in two steps.

from $i = n$ down to 0, we obtain smaller and smaller sets of increasingly de-saturated versions of f , which converge towards $f_0 = f$.

To specify how the set of pairs used for recursion is determined, let there exist a function *super* which takes the index of a tuple from L_i and returns the set of indices from L_{i+1} which denote tuples identical to the first tuple, excluding the last element:

$$\text{super} : \mathbb{N} \times \mathbb{N} \rightarrow \mathcal{P}(\mathbb{N}) :: (i, j) \mapsto \{j' \mid \forall j'' \in [1, k']. [w_1^{j''} = w_1^{j'} \wedge \dots \wedge w_i^{j''} = w_i^{j'}]\} \text{ where } k' = |L_{i+1}|$$

Using this function, the regression set for some f_i^j can be defined as $\{(w_{i+1}^{j'}, f_{i+1}^{j'}) \mid j' = \text{super}(i, j)\}$.

7.2.2 Experiments

We evaluated tensors produced using this multi-step regression method against the datasets presented in Chapter 6, producing three new sets of experimental results. In this section, I will first report the model parameters initially described by Georgiana Dinu, before reporting the experimental results that show this new learning procedure to produce models that perform competitively against other leading models.

Source corpus We extract co-occurrence data from the concatenation of the Web-derived ukWaC corpus², a mid-2009 dump of the English Wikipedia³ and the British National Corpus⁴. The corpus has been tokenised, POS-tagged and lemmatised with the TreeTagger [71] and dependency-parsed with the MaltParser [43]. It contains about 2.8 billion tokens.

Co-occurrence extraction We collect vector representations for the top 8K most frequent nouns and 4K verbs in the corpus, as well as for the subject-verb (320K) and subject-verb-object (1.36M) phrases containing one of the verbs to be used in one of the experiments below and subjects and objects from the list of top 8K nouns. For all target items, we collect within-sentence co-occurrences with the top 10K most frequent content words (nouns, verbs, adjectives and adverbs), save for a stop list of the 300 most frequent words. We extract co-occurrence statistics at the lemma level, ignoring inflectional information.

Weighting Following standard practice, raw co-occurrence counts are transformed into statistically weighted scores. We tested various weighting schemes of the semantic space on a word similarity task. We used a subset of the MEN data-set [7] containing 2000 pairs of words (present in our vocabulary) together with human-assigned similarity judgments obtained through crowdsourcing. We observed that non-negative pointwise mutual information (PMI) and local mutual information (raw frequency count multiplied by PMI score) generally outperform other weighting schemes by a large margin, and that PMI in particular works best when combined with dimensionality reduction by non-negative matrix factorization (described below). Consequently, we pick PMI weighting for our experiments.

Dimensionality reduction Reducing co-occurrence vectors to lower dimensionality is a common step in the construction of distributional semantic models. Extensive evidence suggests that dimensionality reduction does not affect, and might even improve the quality of lexical semantic vectors [55, 70, 72]. In our setting, dimensionality reduction is virtually a necessary step, since working with 10K-dimensional vectors is problematic for the Regression approach, which requires learning matrices and tensors with dimensionalities which are quadratic and cubic in the dimensionality of the input vectors, respectively.

We consider two dimensionality reduction methods, the Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF). SVD is the most common technique in distributional semantics, and it was used by [5]. NMF is a less commonly adopted method, but it has also been shown to be an effective dimensionality reduction technique for

²<http://wacky.sslmit.unibo.it/>

³<http://en.wikipedia.org>

⁴<http://www.natcorp.ox.ac.uk/>

distributional semantics [24]. It has a fundamental advantage from our point of view: the Multiply and Kronecker composition approaches, because of their multiplicative nature, cannot be meaningfully applied to vectors containing negative values. NMF, unlike SVD, produces non-negative vectors, and thus allows a fair comparison of all composition methods in the same reduced space. For this reason, we follow [5] in performing dimensionality reduction to obtain 300-dimensional representations.

We perform the Singular Value Decomposition of the input matrix X : $X = U\Sigma V^T$ and, like Baroni and Zamparelli and many others, pick the first $k = 300$ columns of $U\Sigma$ to obtain reduced representations. Non-negative Matrix Factorization factorizes a $(m \times n)$ non-negative matrix X into two $(m \times k)$ and $(k \times n)$ non-negative matrices: $X \approx WH$ (we normalize the input matrix to $\sum_{i,j} X_{ij} = 1$ before applying NMF). We use the Matlab implementation of the projected gradient algorithm⁵ proposed in [57], which minimizes the squared error of Frobenius norm $F(W, H) = \|X - WH\|_F^2$. We set $k = 300$ and we use W as reduced representation of input matrix X . For both SVD and NMF, the latent dimensions are computed using a “core” matrix containing nouns and verbs only, subsequently projecting phrase vectors onto the same space. In this way, the dimensions of the reduced space do not depend on the ad-hoc choice of phrases required by our experiments.

By experimenting with the MEN data-set of word similarity judgments (see above), we found that the performance using distributional semantic vectors from the original 10K-dimensional space or from the two reduced spaces is very similar, confirming the hypothesis that dimensionality reduction does not have a negative impact on the quality of distributional word representations.

We evaluated this new learning procedure against other models of composition, most of which were discussed in Chapter 6. Here are the ones we used in the new set of experiments.

Verb is a baseline measuring the cosine between the verbs in two sentences as a proxy for sentence similarity (e.g., similarity of *mom sings* and *boy dances* is approximated by the cosine of *sing* and *dance*).

We adopt the widely used and generally successful multiplicative and additive models of [64] and others. Composition with the **Multiply** and **Add** methods is achieved by, respectively, component-wise multiplying and adding the vectors of the constituents of the sentence we want to represent. Vectors are normalised before addition, as this has consistently shown to improve the performance of additive models in earlier experiments run by Baroni and his colleagues.

In §5.3.6, I proposed an alternative implementation of the general DisCoCat approach

⁵Available at <http://www.csie.ntu.edu.tw/~cjlin/nmf/>.

to compositional distributional semantics [19], namely **Kronecker**. Under this approach, a transitive sentence is a matrix S derived from:

$$S = (\vec{v} \otimes \vec{v}) \odot (\overrightarrow{subj} \otimes \overrightarrow{obj})$$

That is, if nouns and verbs live in a n -dimensional space, a transitive sentence is a $n \times n$ matrix given by the component-wise multiplication of two Kronecker products: that of the verb vector with itself and that of the subject and object vectors. I showed, in §6.2, that this method outperforms other implementations of the same formalism and is the current state of the art on the transitive sentence task, which we also tackle below. For intransitive sentences, the same approach reduces to the component-wise multiplication of verb and subject vectors, that is, to the Multiply method.

Training of nouns and verbs under the proposed (multi-step) **Regression** model is implemented using Ridge Regression (RR) [45]. RR, also known as L_2 regularized regression, is a different approach from the Partial Least Square Regression (PLSR) method that was used in previous related work [5, 42] to deal with the multicollinearity problem. When multicollinearity exists, the matrix $X^T X$ (X here is the input matrix after dimensionality reduction) becomes nearly singular and the diagonal elements of $(X^T X)^{-1}$ become quite large, which makes the variance of weights too large. In RR, a positive constant λ is added to the diagonal elements of $X^T X$ to strengthen its non-singularity. Compared with PLSR, RR has a simpler solution for the learned weight matrix $B = (X^T X + \lambda I)^{-1} X^T Y$ and produces competitive results at a faster speed. For each verb matrix or tensor to be learned, we tuned the parameter λ by generalized cross-validation [32]. The objective function used for tuning minimizes least square error when predicting corpus-observed sentence vectors or intermediate VP matrices; the data sets we evaluate the models on are *not* touched during tuning.

The training examples for Regression are constructed by combining the 8K nouns we have vectors for with any verb in the evaluation sets into subject-verb-(object) constructions, and extracting the corresponding vectors from the corpus, where attested (vectors are normalised before feeding them to the regression routine). We use only example vectors with at least 10 dimensions with non-zero basis weights before dimensionality reduction, and we require at least 3 training examples per regression. For the first experiment (intransitives), these (untuned) constraints result in an average of 281 training examples per verb. In the second experiment, in the verb-object matrix estimation phase, we estimate on average 324 distinct matrices per verb, with an average of 15 training examples per matrix. In the verb tensor estimation phase we use all relevant verb-object matrices as training

examples.

Experimental Results We present the results for two experiments testing our set of models in Table 7.1. ‘Humans’ is inter-annotator correlation. The multiplication-based Multiply and Kronecker methods are not well-suited for the SVD space and their performance is reported in NMF space only. ‘Kronecker’ is only defined for the transitive case, with ‘Multiply’ functioning as its intransitive-case equivalent.

We first tested our set of models against the intransitive verb dataset presented in §6.2, to verify that the regression approach developed for adjectives worked suitably for intransitive verbs as well. The results in Table 7.1(a) show that the Regression-based model achieves the best correlation when applied to SVD space, confirming that the approach proposed by Baroni and Zamparelli for adjective-noun constructions can be straightforwardly extended to composition of a verb with its subject with good empirical results. The Regression model also achieves good performance in NMF space, where it is comparable to Multiply. Multiply was found to be the best model by Mitchell and Lapata, and we confirm their results here (recall that Multiply can also be seen as the natural extension of Kronecker to the intransitive setting). The correlations attained by Add and Verb are considerably lower than those of the other methods.

(a) Intransitive Sentences		(b) Transitive Sentences	
<i>method</i>	ρ	<i>method</i>	ρ
Humans	0.40	Humans	0.62
Multiply.nmf	0.19	Regression.nmf	0.29
Regression.nmf	0.18	Kronecker.nmf	0.25
Add.nmf	0.13	Multiply.nmf	0.23
Verb.nmf	0.08	Add.nmf	0.07
Regression.svd	0.23	Verb.nmf	0.04
Add.svd	0.11	Regression.svd	0.32
Verb.svd	0.06	Add.svd	0.12
		Verb.svd	0.08

Table 7.1: Spearman correlation of composition methods with human similarity intuitions on two sentence similarity data sets (all correlations significantly above chance).

We then applied our regression method to the task of learning transitive verbs from a corpus, and tested these representations against the transitive verb dataset presented in §6.2. As the results in Table 7.1(b) show, the Regression model performs very well again, better than any other methods in NMF space, and with a further improvement when SVD is used, similarly to the first experiment. The Kronecker model is also competitive, confirming the results from §6.2. Neither Add nor Verb achieve very good results, although even for them

the correlation with human ratings is significant.

7.2.3 Discussion

The results presented in the previous section show that our iterative linear regression algorithm outperforms the leading multiplicative method on intransitive sentence similarity when using SVD (and it is on par with it when using NMF), and outperforms both the multiplicative method and the leading Kronecker model in predicting transitive sentence similarity.

Here again, we saw that Kronecker also performs very well in our modified experimental setup (although not as well as Regression). The main advantage of Kronecker over Regression lies in its simplicity: there is no training involved, all it takes is two vector outer products and a component-wise multiplication.

While our new regression-based model’s estimation procedure is considerably more involved than for Kronecker, the model has much to recommend it, both from a statistical and from a linguistic point of view. On the statistical side, there are many aspects of the estimation routine that could be tuned on automatically collected training data, thus bringing up the Regression model performance. We could, for example, harvest a larger number of training phrases (not limiting them to those that contain nouns from the 8K most frequent in the corpus, as we did), or *vice versa* limit training to more frequent phrases, whose vectors are presumably of better quality. Moreover, Ridge Regression is only one of many estimation techniques that could be tried to come up with better matrix and tensor weights.

On the linguistic side, the model is clearly motivated as an instantiation of the vector-space “dual” of classic composition by function application via the tensor contraction operation, as discussed earlier in this chapter. Moreover, Regression produces vectors of the same dimensionality for sentences formed with intransitive and transitive verbs, whereas for Kronecker, if the former are n -dimensional vectors, the second are $n \times n$ matrices. Thus, under Kronecker composition, sentences with intransitive and transitive verbs are not directly comparable, which is counter-intuitive (being able to measure the similarity of, say, *kids sing* and *kids sing songs* is both natural and practically useful).

7.3 Further Syntactic Extensions: Combinatory Categorical Grammar

In Chapter 4, I discussed procedures for extending the DisCoCat framework to work with other grammatical formalisms, and showed how Context Free Grammars and Lambek Grammars could be included into the formalism in such a way. In this section, I sketch the foundations for extending DisCoCat to work with a more powerful grammatical formalism called Combinatory Categorical Grammar, a weakly context sensitive grammar which is widely used and comes with efficient parsing tools such as the C&C parser of [22]. I show that at least some aspects of the grammar can be given categorical semantics, and briefly discuss aspects which should be the subject of future work, should we wish to fully integrate variants of CCG into the DisCoCat formalism.

7.3.1 Combinatory Categorical Grammar

Combinatory Categorical Grammars (CCGs) are very similar in spirit—and notation—to Lambek Grammar, discussed in §4.3. They are categorical, in that every word in a natural language belongs to one or more syntactic categories; and they are combinatorial in that syntactic categories act as functions which combine to produce syntactical analyses of phrases. Because I also talk about category theory, I will talk of CCG types instead of CCG categories to avoid confusion.

CCG types are generally defined recursively as follows:

- Let there be a set of atomic types N, S , etc.
- $A \setminus_C B$ is a type if A and B are types.
- A/B is a type if A and B are types.

This completes the type definition. So far, it is very similar to the type definition for Lambek Grammar. However, the backslash \setminus_C has a slightly different meaning than the backslash \setminus_L used for Lambek grammars, as will become clear below. Generally speaking the two notations can be used interchangeably if the combination operations are rewritten appropriately: $A \setminus_L B$ is equivalent to writing $B \setminus_C A$, and vice-versa. These are merely two different notational conventions used in categorial grammars.

CCGs come equipped with a set of reduction rules. The principal ones surveyed by [30], are as follows:

- Application:

$$X/Y, Y \rightarrow X$$

$$Y, X \backslash_C Y \rightarrow X$$

- Composition:

$$X/Y, Y/Z \rightarrow X/Z$$

$$Y \backslash_C Z, X \backslash_C Y \rightarrow X \backslash_C Z$$

- Type raising:

$$X \rightarrow T/(T \backslash_C X)$$

$$X \rightarrow T \backslash_C (T/X)$$

So far, these rules are almost exactly similar to the combination rules for Lambek Grammar presented in §4.3, with the difference that \cdot operations become commas and the expressions of the form $X \backslash_L Y$ are turned into $Y \backslash_C X$ in the rules.

Two sets of rules which are not found in Lambek Grammar are the following:

- Crossed composition:

$$X/Y, Y \backslash_C Z \rightarrow X \backslash_C Z$$

$$Y/Z, X \backslash_C Y \rightarrow X/Z$$

- Substitution:

$$(X/Y)/Z, Y/Z \rightarrow X/Z$$

$$(X/Y) \backslash_C Z, Y \backslash_C Z \rightarrow X \backslash_C Z$$

$$Y/Z, (X \backslash_C Y)/Z \rightarrow X/Z$$

$$Y \backslash_C Z, (X \backslash_C Y) \backslash_C Z \rightarrow X \backslash_C Z$$

Additionally, both composition and crossed composition have a generalised form which allows the rule to be applied when Z is not an atomic type.

These rules are used to perform grammatical analysis of sentences as follows: if there is some type-assignment for each word in the sentence such that the sequence of types

assigned to the sentence reduces to some sentence type S through the combinatorial rules described above, then the sentence is grammatical.

There are many additional rules, restrictions and modifications of the above that are available to generate CCGs with certain properties. For example, CCGs are generally mildly-context sensitive [84], meaning that they can identify or generate sentence structures which context free grammars such as CFGs, LGs or pregroup grammars could not. However, with certain rule restrictions such as removing the unrestricted type raising rule described above, it can be shown [30] that some CCGs are in fact context free.

7.3.2 Categorical Semantics for CCGs

I now turn to the task of trying to represent CCGs as categories. This section will provide foundations for this task and discuss some of the issues faced. It does not aim to provide a complete solution, leaving this for future work.

In §4.3.2, we saw that Lambek Grammars could be viewed as bi-closed monoidal categories. Since CCGs and LGs have some very similar combination rules, let us first consider how far bi-closed monoidal categories get us.

We begin by assuming that the categorical representation **CCG** of a CCG is a bi-closed monoidal category, with the following objects:

- For each atomic type A in the CCG there is some object A in $ob(\mathbf{CCG})$.
- For each type of the form A/B in the CCG there is some object $A \multimap B$ in $ob(\mathbf{CCG})$, where A and B are in $ob(\mathbf{CCG})$.
- For each type of the form $A \setminus_C B$ in the CCG there is some object $B \multimap A$ in $ob(\mathbf{CCG})$, where A and B are in $ob(\mathbf{CCG})$.
- For each sequence of types A, B permitted by the CCG, there is an object $A \otimes B$ in $ob(\mathbf{CCG})$, where A and B are in $ob(\mathbf{CCG})$.

So far, this is effectively the same set of objects as one would expect to find in the categorical representation **LG** of a Lambek Grammar. The same goes for the first set of combination operations. As a reminder, the biclosed category has the following general morphisms for any A, B and C in $ob(\mathbf{CCG})$:

- $ev_{A,B}^l : A \otimes (A \multimap B) \rightarrow B$
- $ev_{A,B}^r : (A \multimap B) \otimes B \rightarrow A$

- $\Lambda^l(f) : C \rightarrow A \multimap B$ for any $f : A \otimes C \rightarrow B$
- $\Lambda^r(g) : C \rightarrow A \multimap B$ for any $g : C \otimes B \rightarrow A$

Using these we can define the first set of composition operations categorically, as was done in §4.3.2:

- **Application operations** are simply the evaluation morphisms.
- **Composition:** for any pair of objects $A \multimap B$ and $B \multimap C$ in the category, there is a morphism

$$\text{comp}_{A \multimap B, B \multimap C}^r : (A \multimap B) \otimes (B \multimap C) \rightarrow A \multimap C$$

Likewise, for any pair of objects $A \multimap B$ and $B \multimap C$ in the category, there is a morphism

$$\text{comp}_{A \multimap B, B \multimap C}^l : (A \multimap B) \otimes (B \multimap C) \rightarrow A \multimap C$$

- **Type raising** morphisms $\text{raise}_{A,B}^l$ and $\text{raise}_{A,B}^r$ are just the right and left currying of the left and right evaluation morphisms:

- $\text{raise}_{A,B}^l = \Lambda^r(\text{ev}_{A,B}^l) : A \rightarrow B \multimap (A \multimap B)$
- $\text{raise}_{A,B}^r = \Lambda^l(\text{ev}_{B,A}^r) : A \rightarrow (B \multimap A) \multimap B$

Note that the composition morphisms make no assumptions about the structure of C in the first case and A in the second, thereby representing both composition and generalised composition at the same time. The diagrams for these operations are shown in Figures 4.3–4.7 of Chapter 4.

So far, it seems like bi-closed monoidal categories are a suitable categorical representation for CCGs. However, a problem comes up when we consider crossed composition and substitution, which have the following forms:

$$\begin{aligned} \text{cross}_{A,B,C}^r &: (A \multimap B) \otimes (C \multimap B) \rightarrow C \multimap A \\ \text{cross}_{A,B,C}^l &: (B \multimap C) \otimes (B \multimap A) \rightarrow A \multimap C \\ \text{lsub}_{A,B,C}^l &: ((A \multimap B) \multimap C) \otimes (B \multimap C) \rightarrow A \multimap C \\ \text{lsub}_{A,B,C}^r &: (C \multimap (A \multimap B)) \otimes (C \multimap B) \rightarrow C \multimap B \\ \text{rsub}_{A,B,C}^l &: (B \multimap C) \otimes ((A \multimap B) \multimap C) \rightarrow A \multimap C \\ \text{rsub}_{A,B,C}^r &: (C \multimap B) \otimes (C \multimap (A \multimap B)) \rightarrow C \multimap A \end{aligned}$$

It should be fairly clear from the domain and codomains of these morphisms that bi-closed monoidal categories will most likely not support these operations using just the evaluation and currying morphisms provided above. This becomes especially clear when we consider the simple diagrammatic form of these morphisms, shown in Figure 7.3 for the *cross* morphisms, and in Figures 7.4 and 7.5 for the *lsub* and *rsub* morphisms, respectively.

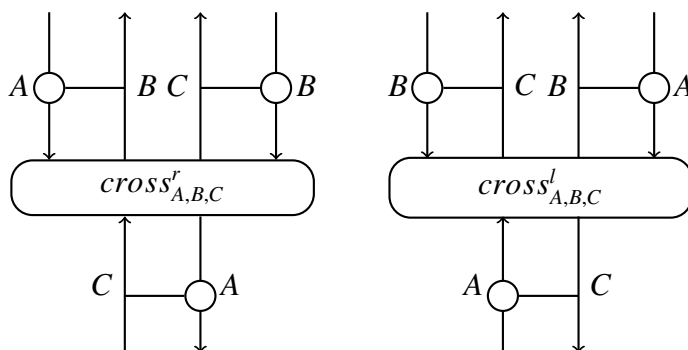


Figure 7.3: Simple diagrammatic form of *cross* morphisms.

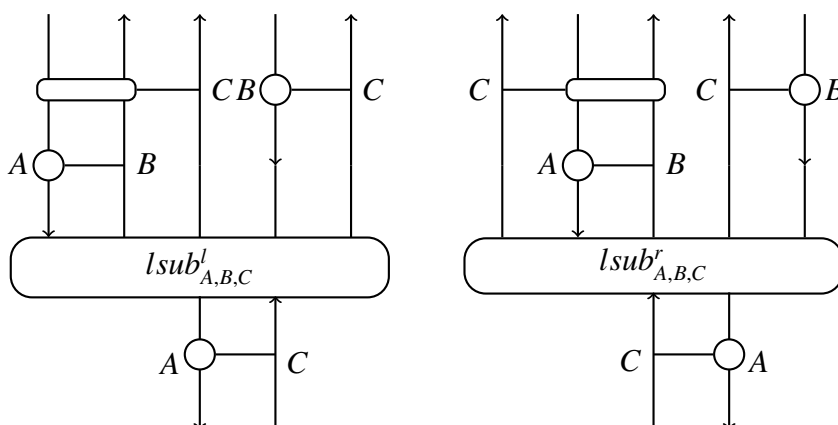


Figure 7.4: Simple diagrammatic form of *lsub* morphisms.

For the *cross* morphisms in Figure 7.3 we can see that within the internal structure of the morphisms, there must be some point where the wires for B and the wire for C cross over each other in order for the two B wires to form a cup, which is not possible in a bi-closed monoidal category. For the substitutions morphisms in Figures 7.4 and 7.5, the problem is a little more subtle: not only must wires cross for the B wires to form a cup, but there are two C wires as input and only one as output. This means that there must be some mechanism within the morphism that takes two wires bearing the same letter and direction, and produces one wire bearing the same letter and direction as the other two.

Let us begin by addressing the issue of crossing wires: we modify our hypothesis about the categorical structure of **CCG** to state that it is a closed monoidal category. Typically

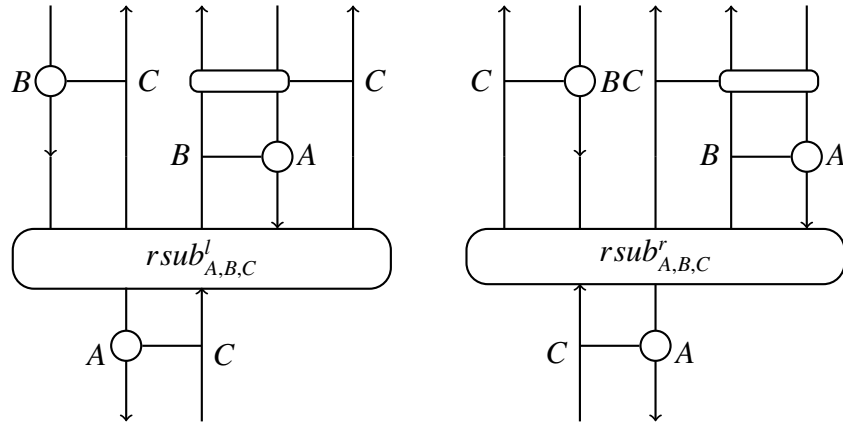


Figure 7.5: Simple diagrammatic form of $rsub$ morphisms.

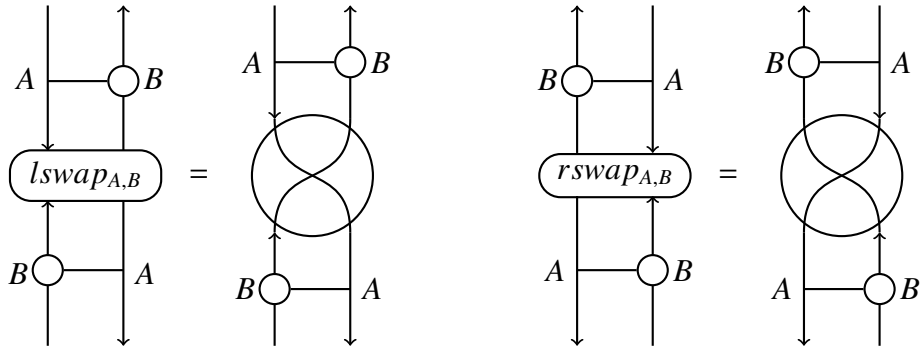


Figure 7.6: Diagrams for $lswap$ and $rswap$ morphisms.

a closed monoidal category is defined identically to a bi-closed monoidal category, except for the fact that instead of the pair of objects $A \multimap B$ and $B \multimap A$ for each A and B in the category, there is a single object $A \multimap B$ which satisfies $A \otimes (A \multimap B) \cong (A \multimap B) \otimes A \cong B$. In short, $A \multimap B$ in the closed monoidal category acts like both $A \multimap B$ and $B \multimap A$ in a bi-closed monoidal category. Here, we will modify our categorical definition explicitly to turn the bi-closed category into a closed category by adding the following morphisms for each A and B :

$$lswap_{A,B} : A \multimap B \rightarrow B \multimap A \quad rswap_{A,B} : B \multimap A \rightarrow A \multimap B$$

such that $rswap_{A,B} \circ lswap_{A,B} = id_{A \multimap B}$ and $lswap_{A,B} \circ rswap_{A,B} = id_{B \multimap A}$ (i.e. $lswap$ and $rswap$ morphisms are isomorphisms). We give these morphisms the diagrammatic representation shown in Figure 7.6. Using these morphisms, we can diagrammatically show the inner structure for the *cross* morphisms initially shown in Figure 7.3, as demonstrated in Figure 7.7.

To deal with the substitution morphisms, I suggest borrowing a particular map from

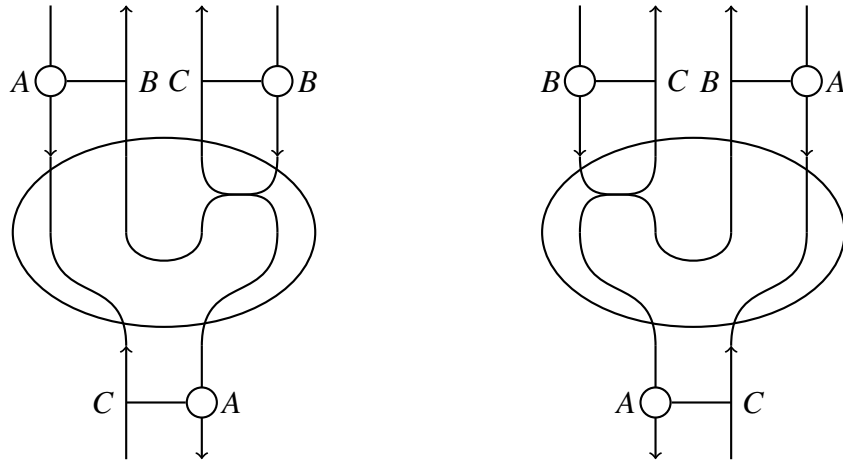


Figure 7.7: Expanded diagrammatic form of *cross* morphisms.

Frobenius Algebras represented in categories as done in [49], defined as follows for any object A in our category:

$$\mu_A : A \otimes A \rightarrow A$$

This map effectively works like a cup, cancelling out some of the information provided as input, but instead of completely eliminating the input information, it preserves some of it and outputs it. Diagrammatically, it is represented as shown in Figure 7.8. Using this morphism, we can diagrammatically show the inner structure for the substitution morphisms initially shown in Figures 7.4 and 7.5, as demonstrated in Figures 7.9 and 7.10.

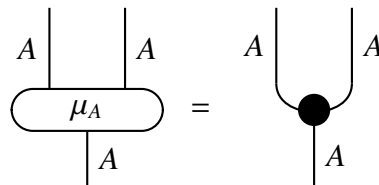


Figure 7.8: Diagrammatic representation of μ morphisms.

So to summarise what we have seen here, I hypothesise that closed monoidal categories with an additional set of μ morphisms model those composition operations in CCGs which we discussed here. I say ‘hypothesise’ for two reasons. Firstly, because the inner structure of the cross composition and substitution morphisms presented here is obtained through ‘diagrammatic reverse-engineering’, rather than through any real appeal to the algebraic structure of CCGs (as there is not a tidy definition of this structure, especially compared to Pregroup Grammars or Lambek Grammars). As such, future work should seek to determine whether or not the inner structure of these morphisms presented here makes sense from an algebraic—or indeed a linguistic—standpoint to confirm or reject this hypothesis.

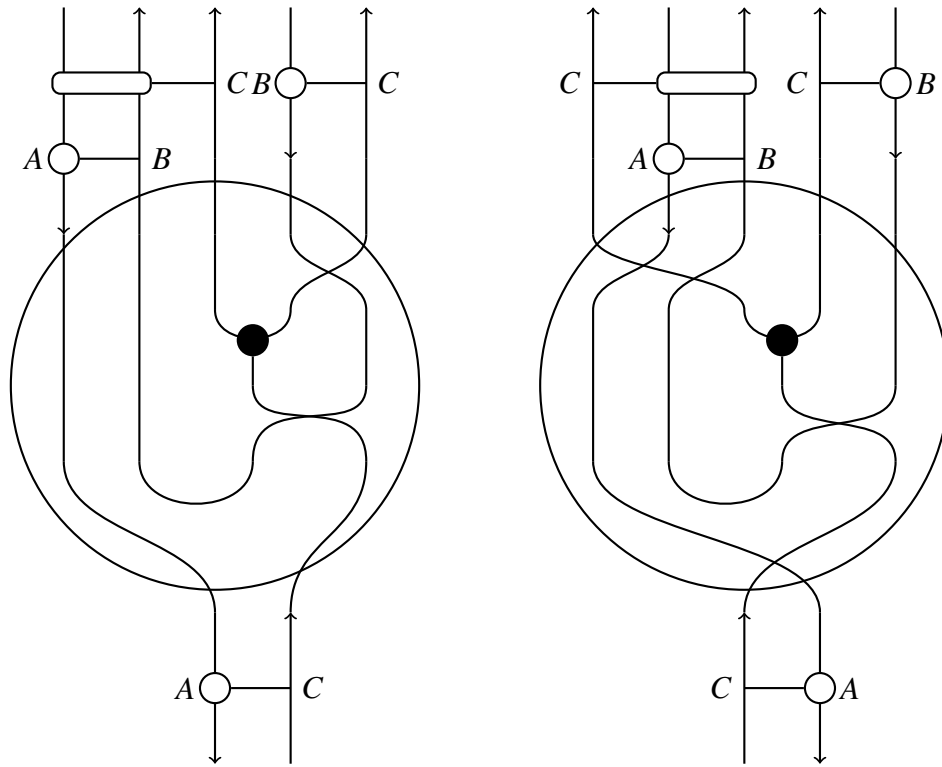


Figure 7.9: Expanded diagrammatic form of *lsub* morphisms.

Secondly, this is a hypothesis because it does not fully capture all the variants of CCGs present in the literature. Earlier in this section, I talked about optional restrictions placed on rules such as composition and cross composition. Other work in the CCG literature (e.g. [4]) goes further in suggesting that the slashes should be augmented with compositional modalities, and that restrictions should apply to most rules based on the modalities of the slashes involved. For both cases, there is no obvious way of modelling restrictions in closed monoidal categories. I leave it to further work to determine whether or not this matters, and if so, how these aspects of CCG can be incorporated into our categorical discourse.

An additional topic for further work on this subject would be to check that no issues arise from the introduction of swap morphisms into this category. In [81, 82], van Benthem shows that associative and commutative extensions of the Lambek calculus underlying Lambek grammar recognises/generates all permutation closures of context-free languages, which makes it unsuitable for syntactic analysis. In practical terms, this means that the grammar will overgenerate, recognising sentences as grammatical which are not meant to be recognised as such. This ‘overgenerative’ property also holds for the categorical representation of CCG presented above, for the case of unrestricted cross-composition morphisms. Insofar as this is an issue for CCG as a whole, rather than the categorical rep-

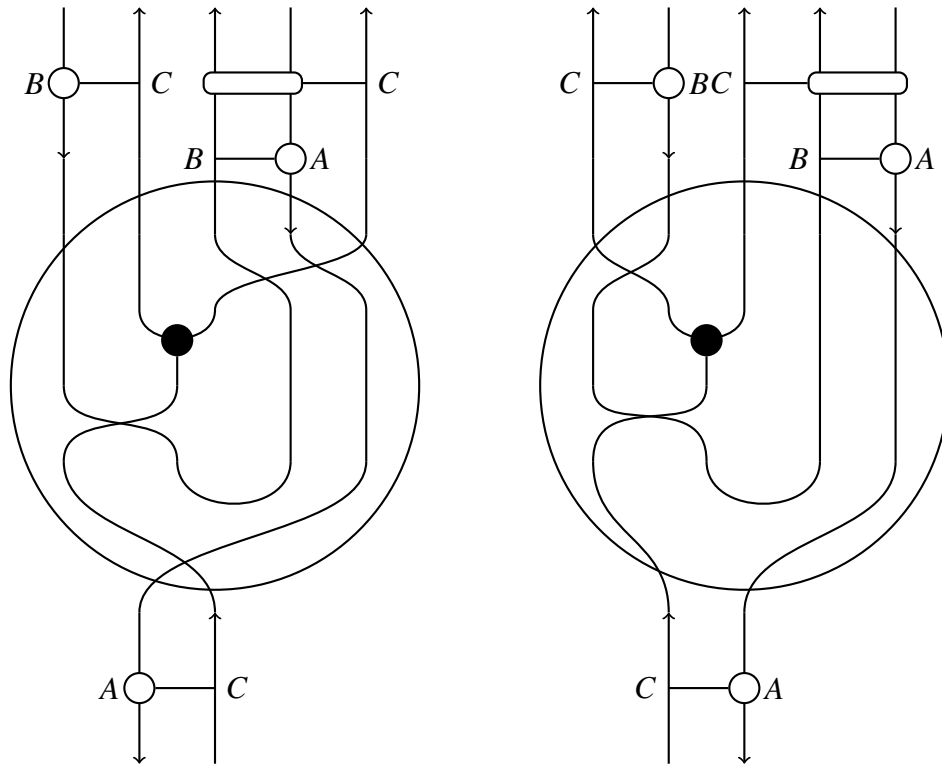


Figure 7.10: Expanded diagrammatic form of *rsub* morphisms.

resentation, I will consider this outside of the scope of this thesis. However, this strongly supports the case for further investigation of how to include rule restrictions into the categorical representation of CCGs.

7.3.3 Defining a Functor

To complete this section, I will briefly present how a functorial passage from the closed monoidal category **CCG** defined here to the category of vector spaces **FVect** can be defined. Because of the similarity between **LG** and **CCG**, we can re-use substantial parts of the work presented in §4.3.3 here.

Let F be a functor between **CCG** and **FVect**. We begin by assigning each atomic type A in the CCG to a vector space $V = F(A)$. We then define the compound objects as was done in **LG**:

$$F(A \otimes B) = F(A) \otimes F(B)$$

$$F(A \multimap B) = F(A) \otimes F(B)$$

$$F(A \multimap B) = F(A) \otimes F(B)$$

The functors mapping those composition operations in CCGs that are also in LGs are exactly as defined for the case of a functor between **LG** and **FVect**, and are as follows:

$$\begin{aligned}
F(\text{ev}_{A,B}^l) &= \epsilon_{F(A)} \otimes 1_{F(B)} : F(A) \otimes F(A) \otimes F(B) \rightarrow F(B) \\
F(\text{ev}_{A,B}^r) &= 1_{F(B)} \otimes \epsilon_{F(A)} : F(B) \otimes F(A) \otimes F(A) \rightarrow F(B) \\
F(\text{comp}_{A,B,C}^r) &= 1_{F(A)} \otimes \epsilon_{F(B)} \otimes 1_{F(C)} : F(A) \otimes F(B) \otimes F(B) \otimes F(C) \rightarrow F(A) \otimes F(C) \\
F(\text{comp}_{A,B,C}^l) &= 1_{F(A)} \otimes \epsilon_{F(B)} \otimes 1_{F(C)} : F(A) \otimes F(B) \otimes F(B) \otimes F(C) \rightarrow F(A) \otimes F(C) \\
F(\text{raise}_{A,B}^l) &= (\eta_{F(B)} \otimes 1_{F(A)}) : F(A) \rightarrow F(B) \otimes F(B) \otimes F(A) \\
F(\text{raise}_{A,B}^r) &= (1_{F(A)} \otimes \eta_{F(B)}) : F(A) \rightarrow F(A) \otimes F(B) \otimes F(B)
\end{aligned}$$

This leaves us only with the task of defining a functorial passage for the cross composition and substitution morphisms. We begin with the internal structure of the cross composition morphisms, which we can read from the diagrams:

$$\begin{aligned}
F(\text{cross}_{A,B,C}^r) &= F(\text{comp}_{A,B,C}^l \circ (1_{A \circ B} \otimes \text{lswap}_{A,B})) \\
&= (1_{F(A)} \otimes \epsilon_{F(B)} \otimes 1_{F(C)}) \circ (1_{F(A)} \otimes 1_{F(B)} \otimes F(\text{lswap}_{A,B})) \\
F(\text{cross}_{A,B,C}^l) &= F(\text{comp}_{A,B,C}^r \circ (\text{rswap}_{A,B} \otimes 1_{B \rightarrow A})) \\
&= (1_{F(A)} \otimes \epsilon_{F(B)} \otimes 1_{F(C)}) \circ (F(\text{rswap}_{A,B}) \otimes 1_{F(A)} \otimes 1_{F(B)})
\end{aligned}$$

We know how the functorial passage for every element of the above expressions above is defined, except for the swap morphisms. To complete the functorial passage for the cross composition morphisms, we must therefore describe what the swap morphisms correspond to in **FVect**. As **FVect** is a symmetric monoidal category, there is an isomorphism $\text{swap}_{A,B}$ for all objects A and B in $\text{ob}(\mathbf{FVect})$ defined as follows:

$$\text{swap}_{A,B} : A \otimes B \rightarrow B \otimes A$$

This is the obvious target for the swap morphisms in **CCG**, and hence:

$$F(\text{lswap}_{A,B}) = F(\text{rswap}_{A,B}) = \text{swap}_{F(A),F(B)}$$

Linear algebraically, this morphism of **FVect** corresponds to the permutation of tensor indices. The simplest and most intuitive case to illustrate this is that of applying $\text{swap}(A, B)$ to a matrix in $A \otimes B$, which yields a matrix in $B \otimes A$ which is the transpose of the original

matrix.

So we can rewrite the functorial definition for the cross composition morphisms as follows:

$$\begin{aligned} F(\text{cross}_{A,B,C}^r) &= (1_{F(A)} \otimes \epsilon_{F(B)} \otimes 1_{F(C)}) \circ (1_{F(A)} \otimes 1_{F(B)} \otimes \text{swap}_{F(A),F(B)}) \\ F(\text{cross}_{A,B,C}^l) &= (1_{F(A)} \otimes \epsilon_{F(B)} \otimes 1_{F(C)}) \circ (\text{swap}_{F(A),F(B)} \otimes 1_{F(A)} \otimes 1_{F(B)}) \end{aligned}$$

With this in mind, we turn to the functorial passage for substitution morphisms. First for the *lsub* morphisms:

$$\begin{aligned} F(\text{lsub}_{A,B,C}^l) &= F(\text{comp}_{A,B,C}^l \circ (1_{A \circ B} \otimes \text{lswap}_{B,C}) \circ (1_{A \circ B} \otimes \mu_C \otimes 1_B) \\ &\quad \circ (1_{(A \circ B) \circ C} \otimes \text{rswap}_{B,C})) \\ &= (1_{F(A)} \otimes \epsilon_{F(B)} \otimes 1_{F(C)}) \circ (1_{F(A)} \otimes 1_{F(B)} \otimes \text{swap}_{F(A),F(B)}) \\ &\quad \circ (1_{F(A)} \otimes 1_{F(B)} \otimes F(\mu_A) \otimes 1_{F(B)}) \circ (1_{F(A)} \otimes 1_{F(B)} \otimes 1_{F(C)} \otimes \text{swap}_{F(B),F(C)}) \\ F(\text{lsub}_{A,B,C}^r) &= F(\text{rswap}_{A,C} \circ \text{comp}_{A,B,C}^l \circ (1_{A \circ B} \otimes \text{lswap}_{B,C}) \circ (1_{A \circ B} \otimes \mu_C \otimes 1_B) \\ &\quad \circ (\text{lswap}_{A \circ B,C} \otimes 1_{C \circ B})) \\ &= \text{swap}_{F(A),F(C)} \circ (1_{F(A)} \otimes \epsilon_{F(B)} \otimes 1_{F(C)}) \circ (1_{F(A)} \otimes 1_{F(B)} \otimes F(\mu_C) \otimes 1_{F(B)}) \\ &\quad \circ (\text{swap}_{F(A) \otimes F(B),F(C)} \otimes 1_{F(C)} \otimes 1_{F(B)}) \end{aligned}$$

Next, for the *rsub* morphisms:

$$\begin{aligned} F(\text{rsub}_{A,B,C}^l) &= F(\text{lswap}_{A,C} \circ \text{comp}_{C,B,A}^r \circ (\text{rswap}_{C,B} \otimes 1_{B \circ A}) \circ (1_B \otimes \mu_C \otimes 1_{B \circ A}) \\ &\quad \circ (1_{B \circ C} \otimes \text{rswap}_{C,B \circ A})) \\ &= \text{swap}_{F(A),F(C)} \circ (1_{F(C)} \otimes \epsilon_{F(B)} \otimes 1_{F(A)}) \circ (\text{swap}_{F(C),F(B)} \otimes 1_{F(B)} \otimes 1_{F(A)}) \\ &\quad \circ (1_{F(B)} \otimes F(\mu_C) \otimes 1_{F(B)} \otimes 1_{F(A)}) \circ (1_{F(B)} \otimes 1_{F(C)} \otimes \text{swap}_{F(C),F(B) \otimes F(A)}) \\ F(\text{rsub}_{A,B,C}^r) &= F(\text{comp}_{C,B,A}^r \circ (\text{rswap}_{C,B} \otimes 1_{B \circ A}) \circ (1_B \otimes \mu_C \otimes 1_{B \circ A}) \\ &\quad \circ (\text{lswap}_{B,C} \otimes 1_{C \circ (B \circ A)})) \\ &= (1_{F(C)} \otimes \epsilon_{F(B)} \otimes 1_{F(A)}) \circ (\text{swap}_{F(C),F(B)} \otimes 1_{F(B)} \otimes 1_{F(A)}) \\ &\quad \circ (1_{F(B)} \otimes F(\mu_C) \otimes 1_{F(B)} \otimes 1_{F(A)}) \circ (\text{swap}_{F(B),F(C)} \otimes 1_{F(C)} \otimes 1_{F(B)} \otimes 1_{F(A)}) \end{aligned}$$

The remaining unknown here is now the functorial passage for μ maps. An interpretation

of such a Frobenius operation is provided in [49], and should suit our needs. In **FVect** there are morphisms μ_A for any object A , defined as follows:

$$\mu_A : A \otimes A \rightarrow A :: \vec{v} \otimes \vec{w} \mapsto \vec{v} \odot \vec{w}$$

So we can define the functorial passage for μ maps in $hom(\mathbf{CCG})$ as follows:

$$F(\mu_A) = \mu_{F(A)}$$

Using this, we can rewrite the functorial passage for $lswap$ morphisms as follows:

$$\begin{aligned} F(lsub_{A,B,C}^l) &= (1_{F(A)} \otimes \epsilon_{F(B)} \otimes 1_{F(C)}) \circ (1_{F(A)} \otimes 1_{F(B)} \otimes swap_{F(A),F(B)}) \\ &\quad \circ (1_{F(A)} \otimes 1_{F(B)} \otimes \mu_{F(A)} \otimes 1_{F(B)}) \circ (1_{F(A)} \otimes 1_{F(B)} \otimes 1_{F(C)} \otimes swap_{F(B),F(C)}) \\ F(lsub_{A,B,C}^r) &= swap_{F(A),F(C)} \circ (1_{F(A)} \otimes \epsilon_{F(B)} \otimes 1_{F(C)}) \circ (1_{F(A)} \otimes 1_{F(B)} \otimes \mu_{F(C)} \otimes 1_{F(B)}) \\ &\quad \circ (swap_{F(A) \otimes F(B),F(C)} \otimes 1_{F(C)} \otimes 1_{F(B)}) \end{aligned}$$

And the functorial passage for $rswap$ morphisms as follows:

$$\begin{aligned} F(rsub_{A,B,C}^l) &= swap_{F(A),F(C)} \circ (1_{F(C)} \otimes \epsilon_{F(B)} \otimes 1_{F(A)}) \circ (swap_{F(C),F(B)} \otimes 1_{F(B)} \otimes 1_{F(A)}) \\ &\quad \circ (1_{F(B)} \otimes \mu_{F(C)} \otimes 1_{F(B)} \otimes 1_{F(A)}) \circ (1_{F(B)} \otimes 1_{F(C)} \otimes swap_{F(C),F(B) \otimes F(A)}) \\ F(rsub_{A,B,C}^r) &= (1_{F(C)} \otimes \epsilon_{F(B)} \otimes 1_{F(A)}) \circ (swap_{F(C),F(B)} \otimes 1_{F(B)} \otimes 1_{F(A)}) \\ &\quad \circ (1_{F(B)} \otimes \mu_{F(C)} \otimes 1_{F(B)} \otimes 1_{F(A)}) \circ (swap_{F(B),F(C)} \otimes 1_{F(C)} \otimes 1_{F(B)} \otimes 1_{F(A)}) \end{aligned}$$

This completes the functor definition, and this section.

7.4 Next Steps

Throughout this chapter, we have seen three areas in which further progress has been established based on the foundations provided by the earlier parts of this thesis. First, I discussed the difficulties behind integrating logical aspects of natural language into the framework, and the problems that arise from our reliance on multilinear maps within the DisCoCat compositional formalism. Second, I discussed ways in which machine learning methods could be integrated into the framework to replace or supplement the learning methods I had initially developed with Mehrnoosh Sardzadeh. Third, I discussed how grammatical

formalisms with a more complex structure than those previously examined, and with different linguistic and grammatical properties, could possibly be integrated into the DisCoCat formalism. To conclude this chapter, I offer some thoughts on how these three areas can form the foundations for future work, and what issues other researchers in the field may wish to examine.

In §7.1, I showed how aspects of formal semantics and predicate logic could be simulated within a tensor-based compositional framework. In some cases, such as logical connectives, the possibility of modelling operations using multilinear maps depended on the shape of the sentence space. While I showed that some sentence spaces (e.g. $S = B_2$) did allow multilinear maps to model connectives, it was not obvious how quantification could be modelled without appealing to non-linear maps. Researchers wishing to develop this aspect of the DisCoCat framework would therefore do well to consider how this obstacle may be surpassed using categorical logic, or by changing the semantic model from **FVect** to some other form of semantic representation which would naturally allow the integration of non-linearity into the formalism.

A related problem is that of how to model logical operations in non-truth-theoretic semantic representations, such as those developed in Chapter 5. While I have no specific suggestions concerning where to begin addressing this problem, it should be fairly clear from my attempt to simulate predicate logic that the nature and structure of the sentence space S will crucially affect the nature of logical operations. It is possible to simulate predicate logic precisely because we can interpret basis elements of the sentence space as truth values; therefore when considering how logical connectives and quantification would operate over other kinds of values, we must first consider how to interpret the elements of the sentence space in linguistically or logically sensible ways. Arguably, what is lacking in our current non-truth-theoretic models is such an interpretation, and therefore researchers interested in further developing this topic may wish to begin by re-thinking what, precisely, it is that we are representing in our sentence vectors.

In §7.2, I discussed joint work performed with Georgiana Dinu, Yao-Zhong Zhang, Mehrnoosh Sadrzadeh and Marco Baroni, in which we showed that sophisticated machine learning methods could be adapted to learn the semantic representations used in the DisCoCat framework from data. As a by-product, this approach also addressed the problem present in the approach presented in Chapter 5 whereby sentences with verbs of different valency as their head would reside in different sentence spaces. Continuing to develop a data-driven machine-learning approach to learning semantic representations should constitute an important future research direction for this formalism, as we move away from the task of simply evaluating it against simple experiments, and towards the task of applying

it to concrete problems such as machine translation evaluation and machine translation itself, paraphrase detection, information retrieval, sentiment analysis, or any other text-based problems which would benefit from including deeper semantic knowledge into solutions, as is being done in a variety of work produced since the writing of this thesis (e.g. [46, 76]).

Finally, in §7.3, I laid down foundations for integrating CCGs into the DisCoCat formalism, with the hope that fully integrating variants of CCGs into the formalism will be taken up as a subject for further work. This is not only useful because of the powerful parsing tools available for this grammatical formalism, but also for the rich linguistic aspects (cf. [78]) underlying this class of grammars. One of the main benefits of the categorical approach behind DisCoCat is that it allows information to pass between syntactic and semantic structures. Investigating how to exploit the expressive nature of CCGs to represent more sophisticated linguistic phenomena in our semantic models should be both interesting and help give us a better idea about exactly what semantic properties we are modelling, and how our semantic representations may need to evolve to cater to the linguistic properties CCGs address.

Chapter 8

Conclusions

To conclude this thesis, let me begin by summarising the work presented throughout this document. In Chapter 2, I provided a critical overview of the some attempts, new and old, to address the problem of compositionality in distributional models of semantics. We observed a tension between the need to produce tractable representations which could easily be learned and compared, and the need to integrate syntactic, relational, or knowledge-based information into both our semantic representations and our compositional process.

In Chapter 3, I reported a recent effort to produce a general framework, DisCoCat, within which syntax and semantics are put into relation in order to produce compositional distributional models of semantics that would take grammatical structure into account both in the semantic representation of words, and into how they are composed to produce representations of sentences. This established the foundation on which the rest of the thesis was built, with the goal of showing that this framework could indeed produce high quality, learnable models of compositional distributional semantics.

In Chapter 4, I presented various syntactic extensions to the DisCoCat framework, allowing us to use a wider variety of grammatical formalisms as part of the creation of compositional distributional models of semantics. I also described a procedure by which grammatical formalisms not described in this document could be integrated into the framework, by giving them a categorical representation and defining a functorial passage between such a representation and the categorical representation of whichever semantic model we choose to use.

In Chapter 5, I presented a learning procedure for the production of semantic representations to be used within the DisCoCat formalism. This procedure allows us to develop concrete models from the abstract framework. I also discussed a reduced representation for semantic objects used in such models, which allows us to efficiently compute sentence representations. Finally, I introduced a variant on this learning procedure for reduced rep-

representations, based on the Kronecker product of lexical vectors, and showed that reduced representations can generally be viewed as embedded within the full representation of semantic relations, leading to a reduction in computational complexity without modifying the mathematical nature of composition operations.

In Chapter 6, I evaluated the concrete models from the previous chapter in the context of three phrase similarity detection experiments, against other leading models discussed earlier in the thesis. I showed that the models produced by the DisCoCat framework match or outperform competing models, and that the difference between DisCoCat models and rivals grows with the complexity of the sentences used in the experiments.

In Chapter 7, I presented three further extensions to the work done in the rest of the thesis. I discussed options for integrating logical operations into distributional compositional semantic models, and outlined the difficulties and obstacles faced by the DisCoCat framework in trying to accommodate such operations. I presented a new machine learning algorithm developed with colleagues, which improved upon certain aspects of the learning procedures presented earlier in this thesis. I also discussed foundations for the inclusion of Combinatory Categorical Grammars into the DisCoCat framework, and surveyed some of the difficulties faced by trying to fully integrate all variants of such grammars into our categorical formalism. I concluded the chapter by providing suggestions as to future directions research on the topic of categorical compositional distributional models of semantics might take.

Throughout this thesis, I have shown that the DisCoCat framework is not a theoretical toy example of how category theory can be applied to problems in linguistics. Indeed, not only can learning procedures be developed to generate concrete models from it, but the category theoretic aspects themselves provide powerful tools when it comes to the expansion of the framework itself. I discussed how such expansions could take place in the form of integrating new syntactic formalisms into the framework, but also suggested that the categorical properties of the framework may provide ways to surpass the limitations of our semantic representations when it comes to modelling logical aspects of language. In discussing both its potential for expansion and showing its ability to admit machine-learning-inspired learning algorithms, I hope to have demonstrated that the categorical approach to linguistics walks the path between theory and practice. On these grounds, I am convinced that it will constitute an important and fascinating area for future research, as others have already noted [47].

Bibliography

- [1] S. Abramsky and B. Coecke. A categorical semantics of quantum protocols. In *Logic in Computer Science, 2004. Proceedings of the 19th Annual IEEE Symposium on*, pages 415–425. IEEE, 2004.
- [2] H. Alshawi, editor. *The Core Language Engine*. MIT Press, 1992.
- [3] J. Baez and M. Stay. Physics, topology, logic, and computation: A rosetta stone. In B. Coecke, editor, *New Structures in Physics*, volume 813 of *Lecture Notes in Physics*. Springer, 2011.
- [4] J. Baldridge and G. J. M. Kruijff. Multi-modal combinatory categorial grammar. In *Proceedings of the tenth conference of the European chapter of the Association for Computational Linguistics-Volume 1*, pages 211–218. Association for Computational Linguistics, 2003.
- [5] M. Baroni and R. Zamparelli. Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1193. Association for Computational Linguistics, 2010.
- [6] N. Bourbaki. *Commutative Algebra: Chapters 1-7*. Springer-Verlag (Berlin and New York), 1989.
- [7] E. Bruni, G. Boleda, M Baroni, and N. K. Tran. Distributional semantics in technicolor. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 136–145, Jeju Island, Korea, July 2012. Association for Computational Linguistics.
- [8] W. Buszkowski. Lambek grammars based on pregroups. *Logical aspects of computational linguistics*, pages 95–109, 2001.
- [9] W. Buszkowski and K. Moroz. Pregroup grammars and context-free grammars. *Computational Algebraic Approaches to Natural Language, Polimetrica*, pages 1–21, 2008.
- [10] M.D. Choi. Completely positive linear maps on complex matrices. *Linear algebra and its applications*, 10(3):285–290, 1975.

- [11] N. Chomsky. Three models for the description of language. *Information Theory, IRE Transactions on*, 2(3):113–124, 1956.
- [12] S. Clark. *Type-Driven Syntax and Semantics for Composing Meaning Vectors*. 2013.
- [13] S. Clark, B. Coecke, and M. Sadrzadeh. A compositional distributional model of meaning. In *Proceedings of the Second Quantum Interaction Symposium (QI-2008)*. College Publications, 2008.
- [14] S. Clark and S. Pulman. Combining symbolic and distributional models of meaning. In *AAAI Spring Symposium on Quantum Interaction*, 2006.
- [15] J. Cocke. Programming languages and their compilers: Preliminary notes. 1969.
- [16] B. Coecke. Kindergarten quantum mechanics: Lecture notes. In *AIP Conference Proceedings*, volume 810, 2006.
- [17] B. Coecke, E. Grefenstette, and M. Sadrzadeh. Lambek vs. lambek: Functorial vector space semantics and string diagrams for lambek calculus. *Annals of Pure and Applied Logic*, 2013.
- [18] B. Coecke and É.O. Paquette. Categories for the practising physicist. *Arxiv preprint arXiv:0905.3010*, 2009.
- [19] B. Coecke, M. Sadrzadeh, and S. Clark. Mathematical Foundations for a Compositional Distributional Model of Meaning. March 2010.
- [20] J. Curran, S. Clark, and J. Bos. Linguistically motivated large-scale nlp with c&c and boxer. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 33–36, Prague, Czech Republic, June 2007. Association for Computational Linguistics.
- [21] J. R. Curran. *From distributional to semantic similarity*. PhD thesis, 2004.
- [22] J.R. Curran, S. Clark, and J. Bos. Linguistically motivated large-scale nlp with c&c and boxer. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 33–36. Association for Computational Linguistics, 2007.
- [23] J. Dean and S. Ghemawat. MapReduce:simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
- [24] G. Dinu and M. Lapata. Measuring distributional similarity in context. In *Proceedings of EMNLP*, pages 1162–1172, Cambridge, MA, 2010.
- [25] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

- [26] K. Erk and S. Padó. A structured vector space model for word meaning in context. *Proceedings of the Conference on Empirical Methods in Natural Language Processing - EMNLP '08*, (October):897, 2008.
- [27] L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414. ACM, 2001.
- [28] J. R. Firth. A synopsis of linguistic theory 1930-1955. *Studies in linguistic analysis*, 1957.
- [29] T. A. D. Fowler. Parsing ccgbank with the lambek calculus. In *Parsing with Categorical Grammars Workshop ESSLLI 2009 Bordeaux, France Book of Abstracts*, 2009.
- [30] T. A. D. Fowler and G. Penn. Accurate context-free parsing with combinatory categorical grammar. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 335–344. Association for Computational Linguistics, 2010.
- [31] G. Frege. Über sinn und bedeutung. *Zeitschrift für Philosophie und philosophische Kritik*, 100(1):25–50, 1892.
- [32] G. H. Golub, M. Heath, and G. Wahba. Generalized cross-validation as a method for choosing a good Ridge parameter. *Technometrics*, pages 215–223, 1979.
- [33] E. Grefenstette. Analysing Document Similarity Measures. Master’s thesis, University of Oxford, September 2009.
- [34] E. Grefenstette. Towards a formal distributional semantics: Simulating logical calculi with tensors. *Proceedings of the Second Joint Conference on Lexical and Computational Semantics*, 2013.
- [35] E. Grefenstette, G. Dinu, Y. Zhang, M. Sadrzadeh, and M. Baroni. Multi-step regression learning for compositional distributional semantics. *Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013)*, 2013.
- [36] E. Grefenstette and M. Sadrzadeh. Experimental support for a categorical compositional distributional model of meaning. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, 2011.
- [37] E. Grefenstette and M. Sadrzadeh. Experimenting with transitive verbs in a disccat. In *Proceedings of the 2011 EMNLP Workshop on Geometric Models of Natural Language Semantics*, 2011.
- [38] E. Grefenstette, M. Sadrzadeh, S. Clark, B. Coecke, and S. Pulman. Concrete sentence spaces for compositional distributional models of meaning. In *Proceedings of the Ninth International Conference on Computational Semantics*, pages 125–134. Association for Computational Linguistics, 2011.

- [39] E. Grefenstette, M. Sadrzadeh, B. Coecke, S. Pulman, and S. Clark. Concrete Compositional Sentence Spaces. *ESSLLI'10 Workshop on Compositionality and Distributional Semantic Models*, 2010.
- [40] G. Grefenstette. Use of syntactic context to produce term association lists for text retrieval. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 89–97. ACM, 1992.
- [41] G. Grefenstette. *Explorations in automatic thesaurus discovery*. 1994.
- [42] E. R. Guevara. Modelling Adjective-Noun Compositionality by Regression. *ESSLLI'10 Workshop on Compositionality and Distributional Semantic Models*, 2010.
- [43] J. Hall. *MaltParser: An Architecture for Labeled Inductive Dependency Parsing*. Licentiate thesis, Växjö University, Växjö, Sweden, 2006.
- [44] Z. S. Harris. *Mathematical structures of language*. Wiley, 1968.
- [45] T. Hastie, T. Tibshirani, and J. Friedman. *The Elements of Statistical Learning, 2nd ed.* Springer, New York, 2009.
- [46] K.M. Hermann and P. Blunsom. The role of syntax in vector space models of compositional semantics. *Proceedings of ACL, Sofia, Bulgaria, August. Association for Computational Linguistics*, 2013.
- [47] C. Heunen, M. Sadrzadeh, and E. Grefenstette, editors. *Quantum Physics and Linguistics: A Compositional, Diagrammatic Discourse*. Oxford University Press, 2013.
- [48] R. Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the american mathematical society*, 146:29–60, 1969.
- [49] D. Kartsaklis, M. Sadrzadeh, and S. Pulman. A unified sentence space for categorical distributional-compositional semantics: Theory and experiments. In *Proceedings of 24th International Conference on Computational Linguistics (COLING 2012): Posters*, pages 549–558, Mumbai, India, December 2012.
- [50] T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical report, Air Force Cambridge Research Laboratory, 1965.
- [51] J. Lambek. The Mathematics of Sentence Structure. *The American Mathematical Monthly*, 65(3):154 – 170, 1958.
- [52] J. Lambek. Type grammar revisited. *Logical aspects of computational linguistics*, 1999.
- [53] J. Lambek. *From word to sentence. A computational algebraic approach to grammar*. Polimetrica, 2008.
- [54] J. Lambek. *Compact Monoidal Categories from Linguistics to Physics*, pages 451–469. 2010.

- [55] T. K. Landauer and S. T. Dumais. A solution to Plato’s problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 1997.
- [56] J. Lee. *Riemannian manifolds: An introduction to curvature*, volume 176. Springer Verlag, 1997.
- [57] C. Lin. Projected gradient methods for nonnegative matrix factorization. *Neural Computation*, 19(10):2756–2779, 2007.
- [58] S. Mac Lane. *Categories for the working mathematician*. Springer verlag, 1998.
- [59] C. D. Manning, P. Raghavan, and H.dum Schütze. An introduction to information retrieval. *dspace.cusat.ac.in*, 2009.
- [60] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.
- [61] R. Milner. A theory of type polymorphism in programming. *Journal of computer and system sciences*, 17(3):348–375, 1978.
- [62] G. Minnen, J. Carroll, and D. Pearce. Applied morphological processing of english. *Natural Language Engineering*, 7(03):207–223, 2001.
- [63] J. Mitchell and M. Lapata. Vector-based models of semantic composition. In *Proceedings of ACL*, volume 8, 2008.
- [64] J. Mitchell and M. Lapata. Composition in Distributional Models of Semantics. *Cognitive Science*, 2010.
- [65] R. Montague. English as a Formal Language. *Formal Semantics: The Essential Readings*, 1974.
- [66] M. Moortgat. 2 categorial type logics. *Handbook of logic and language*, page 95, 2010.
- [67] T. A. Plate. Holographic reduced representations: Convolution algebra for compositional distributed representations. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 30–35. Citeseer, 1991.
- [68] A. Preller and M. Sadrzadeh. Bell states and negative sentences in the distributed model of meaning. In P. Selinger B. Coecke, P. Panangaden, editor, *Electronic Notes in Theoretical Computer Science, Proceedings of the 6th QPL Workshop on Quantum Physics and Logic*. University of Oxford, 2010.
- [69] M. Sadrzadeh. High-level quantum structures in linguistics and multi-agent systems. In *AAAI Spring symposium on quantum interactions*, 2007.
- [70] M. Sahlgren. *The Word-Space Model*. Dissertation, Stockholm University, 2006.

- [71] H. Schmid. Improvements in part-of-speech tagging with an application to German. In *Proceedings of the EACL-SIGDAT Workshop*, Dublin, Ireland, 1995.
- [72] H. Schütze. *Ambiguity Resolution in Natural Language Learning*. CSLI, Stanford, CA, 1997.
- [73] P. Selinger. A survey of graphical languages for monoidal categories. *New Structures for Physics*, pages 275–337, 2010.
- [74] P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216, 1990.
- [75] P. Smolensky and G. Legendre. *The Harmonic Mind: From Neural Computation to Optimality-Theoretic Grammar Volume I: Cognitive Architecture*. 2006.
- [76] R. Socher, B. Huval, C. D. Manning, and A.Y Ng. Semantic compositionality through recursive matrix-vector spaces. *Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing*, pages 1201–1211, 2012.
- [77] M. Steedman. *The syntactic process*. The MIT press, 2001.
- [78] M. Steedman and J. Baldridge. Combinatory categorial grammar. *Non-Transformational Syntax*, 2009.
- [79] A. Stolcke. Srilm—an extensible language modeling toolkit. In *Seventh International Conference on Spoken Language Processing*, 2002.
- [80] P. D. Turney and P. Pantel. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, 37(1):141–188, 2010.
- [81] J. van Benthem. The semantics of variety in categorial grammar. *Categorial grammar*, 25:37–55, 1988.
- [82] J. van Benthem. *Language in action: Categories, lambdas, and dynamic logic*. MIT Press, 1995.
- [83] C. J. Van Rijsbergen. *The geometry of information retrieval*. Cambridge Univ Pr, 2004.
- [84] K. Vijay-Shanker and D. J. Weir. The equivalence of four extensions of context-free grammars. *Theory of Computing Systems*, 27(6):511–546, 1994.
- [85] R. F. Walters. *Categories and computer science*. Cambridge Univ Press, 1991.
- [86] D. Widdows. *Geometry and Meaning*. University of Chicago Press, 2005.
- [87] D. Widdows. Semantic vector products: Some initial investigations. In *Proceedings of the Second Quantum Interaction Symposium (QI-2008)*. College Publications. Citeseer, 2008.

- [88] L. Wittgenstein. *Philosophical investigations*. Blackwell, 1953.
- [89] D. H. Younger. Recognition and parsing of context-free languages in time n^3 . *Information and control*, 10(2):189–208, 1967.
- [90] F. M. Zanzotto, I. Korkontzelos, F. Fallucchi, and S. Manandhar. Estimating linear models for compositional distributional semantics. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1263–1271. Association for Computational Linguistics, 2010.
- [91] K. Życzkowski and I. Bengtsson. On duality between quantum maps and quantum states. *Open systems & information dynamics*, 11(01):3–42, 2004.