LEARNING OPEN DOMAIN KNOWLEDGE FROM TEXT

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Gábor György Angeli

April 2016

Preface

This thesis tells you all you need to know about...

Acknowledgements

Contents

List of Tables

List of Figures

Chapter 1

Natural Logic

A recurring theme throughout this thesis will be the use of natural logics for natural language tasks. To begin, we should briefly motivate the use of any logical formalism at all for natural language. A wide range of successful approaches in NLP neither use nor need logical reasoning to perform their task: statistical parsing, part-of-speech tagging, document classification, etc. are all fairly agnostic to any logical phenomena in the documents they are operating over. However, as the field of NLP moves away from syntax and shallow classification tasks and increasingly towards natural language *understanding*, we are faced with a necessity to understand meaning of sentences, and understand the implications that can be drawn from them.

Taking some simple examples across a few tasks: in relation extraction, it would be incorrect to assume that Obama was born in Kenya from a sentence *Obama was not born in Kenya*. For sentiment analysis, the sentence *The movie's box office failure should not be taken as indicative of its quality* should get positive sentiment. In question answering, returning *Harrison Ford* for a query asking for all the female stars of Star Wars would be incorrect.

In all of these cases, a system must either explicitly or implicitly learn how to handle these sorts of logical subtleties. Implicitly, a neural network or sufficiently powerful statistical classifier could learn how to treat these queries, at the expense of sufficiently large amount of training data (in the extreme, one can always simply memorizing the logic). Explicitly, a first order theorem prover or Markov Logic network could capture the logical content of a sentence, at the expense of a drop in expresivity and computational efficiency. In this chapter we'll motivate natural logics as an optimal midway between offloading this sort of logical reasoning entirely to statistical / neural methods, and enforcing a rigid logical framework on the problem.

Broadly, the aim of natural logics are to capture a subset of valid logical inferences by appealing directly to the structure of language, as opposed to running deduction in an abstract logical language (e.g., well-formed first-order formulae composed predicates, variables, connectives, and quantifiers). That is to say, a natural logic is primarily a *proof theory*, where the deductive system operates over natural language syntax. To illustrate with an example, we can consider a simple premise:

The cat ate a mouse.

From this, we can run the following logical derivation, using the syntax of natural language as the proof language:

1	*The cat ate a mouse.*
2	*The cat ate a rodent.*
3	*The cat ate an animal.*
4	*The feline ate an animal.*
5	*The carnivore ate an animal.*
6	$\neg$ *No carnivore ate an animal.*

That is to say, if the cat ate a mouse, it is false that no carnivore ate an animal, and this is by virtue of the "proof" presented above. Although it's not unnatural to use a line of reasoning like the one above to justify a conclusion informally, it may be strange the perspective of propositional and first-order logics to treat this as a formal proof. Nonetheless, it is not an altogether unfamiliar sort of reasoning. Natural logics can trace their origins to the syllogistic reasoning of Aristotle. For instance, the proof above is similar to a simple syllogism:

1	*All cats eat mice.*
2	*All cats are carnivores.*
3	*All carnivores eat mice.*

We can furthermore chain these syllogisms in much the same way you would chain a first-order proof. To introduce a motivating example, an Athenian philosopher might retort with the following if a Persian invader claims that all heroes are Persians:

Clearly you are wrong. You see, all Gods live on Mount Olympus. Some heroes are Gods. And, of course, no one who lives on Mount Olympus is Persian.

Sadly our poor philosopher will have likely perished by this point – lest a Greek hero saves him, in which case then there would remain neither a need nor an audience for his argument. But but I can still convince you, the reader, that this was a valid retort by constructing a first-order proof, given in Figure 1.1. The proof follows a proof by contradiction, showing that the hypothesized hero-God (*"some heroes are Gods"*) would have to both live on Olympus and be Persian – a contradiction with our premise: *"no one who lives on Mount Olympus is Persian."*

To contrast with the first order logic proof, there also exists a natural logic proof of our contradiction, based entirely on Aristotilean syllogisms. This proof makes use of two syllogistic patterns chained together, and one of the axiomatic negations:

1	*All Gods live on Mount Olympus*	
2	*Some heroes are Gods*	
3	*Nobody who lives on Mount Olympus is Persian*	
4	*Some heroes live on Mount Olympus*	AII (Darii), 1, 2
5	*Some heroes are not Persian*	EIO (Ferio), 4, 3
6	$\neg$ *All heroes are Persian*	SaP $\perp$ SoP, 5

1	$\forall x\ \text{God}(x) \supset \text{LivesOnOlympus}(x)$	
2	$\exists x\ \text{Hero}(x) \wedge \text{God}(x)$	
3	$\neg\exists x\ \text{LivesOnOlympus}(x) \wedge \text{Persian}(x)$	
4	$\mid\ \forall x\ \text{Hero(x)} \supset \text{Persian}(x)$	
5	$\mid\ a \mid\ \text{Hero}(a) \wedge \text{God}(a)$	$\exists$E, 2
6	$\mid\quad \mid\ \text{Hero}(a)$	$\wedge$E, 5
7	$\mid\quad \mid\ \text{Hero(a)} \supset \text{Persian}(a)$	$\forall$E, 4
8	$\mid\quad \mid\ \text{Persian}(a)$	$\Rightarrow$E, 6, 7
9	$\mid\quad \mid\ \text{God}(a)$	$\wedge$E, 5
10	$\mid\quad \mid\ \text{God}(a) \supset \text{LivesOnOlympus}(a)$	$\forall$E, 1
11	$\mid\quad \mid\ \text{LivesOnOlympus}(a)$	$\Rightarrow$E, 9, 10
12	$\mid\quad \mid\ \text{LivesOnOlympus}(a) \wedge \text{Persian}(a)$	$\wedge$I, 8, 11
13	$\mid\quad \mid\ \exists x\ \text{LivesOnOlympus}(x) \wedge \text{Persian}(x)$	$\exists$I, 12
14	$\mid\ \exists x\ \text{LivesOnOlympus}(x) \wedge \text{Persian}(x)$	R, 12
15	$\mid\ \perp$	$\perp$ I, 3, 14
16	$\neg\ \forall x\ \text{Hero(x)} \supset \text{Persian}(x)$	$\neg$ I, 4—15

Figure 1.1: A Fitch-style first order logic proof refuting *"all heroes are Persian"* given the premises *"All Gods live on Mount Olympus," "Some heroes are Gods,"* and *"No one who lives on Mount Olympus is Persian."*. The proof follows a proof by contradiction (lines 4–15), hinging on showing that the hero-God hypothesized in premise 2, and instantiated as a on line 5, would have to be a Persian who lives on Olympus. This would contradict premise 3.

This example – and in particular the contrast between the two proof approaches above – provides an excellent context for motivating why the enterprise of natural logics is worthwhile, and why research in the area can have a large impact on natural language processing. I'll highlight three concrete motivations in more detail: (1) natural logics are easy to parse into, (2) this parsing is, in a sense, "lossless," and (3) the proofs have the potential to be much more efficient than first order approaches.

Natural logics are easy to parse into. When performing inference in propositional or first order logic, the premises are no longer in natural language, but rather have been parsed into a special logical form. Ignoring the aesthetics of the logical form itself, this mapping is by no means trivial; in fact, an entire subfield of NLP – semantic parsing – focuses exclusively on this problem [2, 9, 10, 17]. Even in our simple example, an intuitively simple utterance like *"some heroes are Gods"* parses into a premise which reads most naturally as *"there exists something which is a hero and a God."* Even more subtly, consider the difference in form for the following two translations:

Sentence	**Logical Form**
Everyone on Mount Olympus is Persian	$\forall x$ LivesOnOlympus$(x) \supset$ Persian(x)
No one on Mount Olympus is Persian	$\neg\exists x$ LivesOnOlympus$(x) \wedge$ Persian(x)

Despite the lexical forms being nearly identical, the logical form has an entirely different structure, changing not only the quantifier but also the connective ($\supset$ versus $\wedge$). By contrast, "parsing to a logical form" in syllogistic logic is nonexistent, and in the more sophisticated natural logics later in this chapter generally reduces to a shallow syntactic parse of the sentence.

Natural logic parsing is "lossless." In general, natural language has more semantic (not to mention pragmatic) content than the logical propositions we extract from it. This is evident in the fact that it's necessary to retrain semantic parsers for new tasks, and underlies the research agenda of defining general semantic representations – e.g., the Abstract Meaning

Representation [1]. Despite being picked to be as concise as possible, even our motivating example has hints of this. The derivation in Figure 1.1 has defined the predicates necessary for our particular derivation (LivesOnOlympus, God, Hero, Persian); but, these are not the only things that could be extracted from the sentences. Semantically, we could extract that Gods are alive. Pragmatically, we should extract that there exist heroes.

To contrast, by operating over natural language directly, the limitations on what natural logics can infer are entirely due to the limitations of the inference mechanism, rather than limitations in what information was successfully extracted from the sentence.

Natural logic proof are efficient. Annecdotally, it should be clear that the proof in Figure 1.1 is longer and more nuanced than the corresponding syllogistic proof. In fact, to a first approximation, one could make a syllogistic theorem prover with nothing more than regular expressions and a small search algorithm. Of course, more expressive natural logics have more difficult search problems; but, nonetheless, they remain in the same spirit of searching over lexical mutations rather than applying symbolic rule sets.

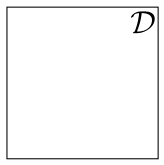
Contrasting with approaches like Markov Logic Networks makes this difference especially salient [14]. Formulating our motivating example as a Markov Logic Network requires outright instantiating the denotations of our predicates on a closed world. That is, we must first enumerate all the heroes, Gods, and residents of Olympus. Thereafter, the task of grounding our formulae on this world and running heavyweight statistical inference algorithm the resulting Markov Network. Making this grounding and inference process tractable is in itself an active area of research [13, 18]. By contrast, natural logics are almost by definition a *proof-theoretic* logic: quantifiers can be reasoned over efficiently using only lexical mutation based proofs. Although they have a model-theoretic interpretation, actually simulating this model is unnecessary and would be impractical.

Of course, despite these advantages it would be unreasonable to advocate for syllogistic reasoning as a practical formalism for modern AI applications. Syllogisms don't allow for compositionality, and are generally restrictive in the range of inferences they warrant. This thesis instead adopts variants of *monotonicity calculus* [15, 16] – a more general-purpose logic which handles a wide range of common phenomena in human language, while nonetheless still operating over the syntax of the language itself.

The remainder of chapter will review denotational semantics (Section 1.1) as a prelude to introducing the monotonicity calculus of Sánchez Valencia [15], described in detail in Section 1.2. From here on, *natural logic* will be used to refer to the monotonicity calculus described in Sánchez Valencia [15]. These sections are intended to give a *model-theoretic* interpretation of the logic that can then be used to justify the potentially ad-hoc seeming proof theory described here. We motivate further research into related formalisms in Section 1.3.

1.1 Denotations

An underlying central concept behind monotonicity calculus (the natural logic used throughout this thesis) is the notion that lexical items can be interpreted in terms of their effect on the set of objects in the world. To be precise, let's introduce a domain $\mathcal{D}$, representing the set of all items and concepts in the world. We'll show this $\mathcal{D}$ as an empty box:



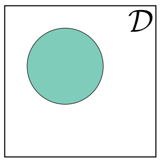
Now, we can start labeling items in this set. For example, this thesis is an item in the world; your hand is likewise an item in the world. We can write these statements as: *this thesis*$\in \mathcal{D}$ and *your hand*$\in \mathcal{D}$; visually, we might show the following:

$\mathcal{D}$
this thesis
$\mathcal{D}$
your hand

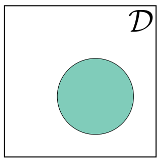
The central thesis behind denotational semantics is the notion that words have *denotations*, which are sets of elements in a particular domain. For example, this thesis and your hand have denotations which are the singleton sets of elements in the domain of *things in the world*. Analogously, cats will be the set of cats in the world, *run* will be the set of actions which we'd label as running, and so forth. The remainder of this section will go over how we map different lexical items to denotations in different domains. This will form the basis of the model theory behind monotonicity calculus, introduced in Section 1.2.

1.1.1 Nouns and Adjectives are Sets

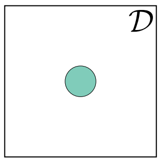
We represent nouns and adjectives as sets – more precisely, as subsets of our domain $\mathcal{D}$. For example, the word *cat* refers to the set of all cats in the world. *Cute* refers to the set of all cute things in the world. Note that this definition subsumes the cases in the previous section: *this thesis* becomes simply the singleton set containing this thesis. We represent these denotations as $[\![cat]\!] \subseteq \mathcal{D}$ (the set of all cats), $[\![cute]\!] \subseteq \mathcal{D}$ (the set of all cute things), etc. Visually, we can show these as a subset of our domain $\mathcal{D}$:



$[\![cute]\!]$



$[\![cat]\!]$

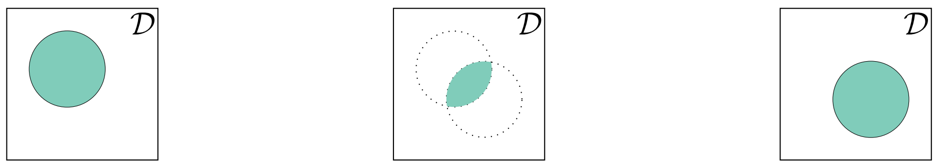


$[\![this\ thesis]\!]$

This is the same sort of representation used in other areas of NLP, most prominently the semantic parsing literature (see, e.g., Liang and Potts [11]). This similarity becomes more clear if we consider nouns and adjectives as *predicates* rather than sets. That is, the word *cat* is a predicate which is true if its argument is a cat, and false otherwise. These two interpretations are, of course, equivalent. A predicate can be represented as the set of entities in the world which satisfy it, and a set of entities in the world can be represented as the predicate that selects them.

The key difference between denotations in monotonicity calculus and semantic parsing is that in the later, the world $\mathcal{D}$ and the denotations of words have concrete (domain-specific) instantiations. For example, $\mathcal{D}$ may be a database of places, people, etc. that can be queried against. In that case, the word *river* would correspond to a finite set of database rows for the rivers in the world. In monotonicity calculus, as we'll see more clearly in Section 1.2, we will never appeal to the explicit denotations of these lexical items. It is sufficient to know that there exist a set of rivers in the world; we never need to explicitly enumerate them.

A natural next question is how we compose lexical items into compound nouns and noun phrases. Like nouns and adjective, noun phrases are denoted as sets of object: *cute cat* is simply the subset of items in the world which are cute cats. Naïvely, this composition amounts to a simple set intersection: *cute cat* refers to the set of things which are both cats and cute; that is, $[\![cute\ cat]\!] = [\![cute]\!] \cap [\![cat]\!]$:



$[\![cute]\!]$ $[\![cute\ cat]\!]$ $[\![cat]\!]$

Less naïvely, we could consider other types of composition (for more information, see e.g., Kamp [8]). For instance, not all adjectives behave intersectively in the way shown above. A *quick DMV visit* is still a DMV visit, but is likely not in the denotation of quick things. Similarly, a *small planet* is nonetheless not generally considered a *small* thing. We refer to these as *subsective* adjectives – the denotation of the compound is a subset of the denotation of the noun, but not a subset of the denotation of the adjective. Another class of adjectives are outright non-subsective. A pseudo-science is not a science; a fake gun is not a gun. In idiomatic expressions, the denotation of the compound has nothing to do with the denotations of the components: a red herring is neither red nor a herring.

We'll revisit compositionality in Section 1.2 and show that handling these sorts of phenomena is important to ensure that the logic remains sound. However, for now we'll continue to review the basic components of the logic.

1.1.2 Sentences are Truth Values

Like most logics, the primary goal of natural logic is to assign truth values to sentences. In natural logic the model-theoretic interpretation of a sentence is simply its truth value. That is to say, sentences can have one of two denotations: they are either true, or they are false.

To make this more formal, and to lay the notational groundwork for the rest of this chapter, let us redefine our domain $\mathcal{D}$ to be $\mathcal{D}_e$ – the domain of entities in the world. So, now, $[\![cat]\!] \subseteq \mathcal{D}_e$. We are now free to specify other domain for other types of lexical items. In our case, let us define $\mathcal{D}_t$ as the domain of truth values. The natural way to define $\mathcal{D}_t$ would be to say: $\mathcal{D}_t = \{true, false\}$ (equivalently, and as we'll find useful later, $\mathcal{D}_t = \{1, 0\}$). The denotation of a sentence is an element in $\mathcal{D}_t$:

$$\begin{aligned} [\![\textit{cats eat mice}]\!] &= true \in \mathcal{D}_t \\ [\![\textit{cats eat carrots}]\!] &= false \in \mathcal{D}_t \end{aligned}$$

1.1.3 Other Lexical Items are Functions

Our review of possible linguistic phenomena or types of lexical items is by no means exhaustive. For instance, we have not covered verbs, or adverbs. Without belaboring the chapter with pedantic thoroughness, the claim of natural logic is that we can construct a denotation for any of these items inductively from the two basic domain – the domain of entities $\mathcal{D}_e$ and the domain of truth values $\mathcal{D}_t$ – and functions based around these domains.

To begin, let's define a set $\mathbf{D}_e$ to be the power set $\mathcal{D}_e$: the set of all subsets of items in the world. This is the set of all possible denotations. We can then define a class of functions $f : \mathbf{D}_e \rightarrow \mathcal{D}_t$. That is, the class of functions mapping from an entity to a truth value. This corresponds most intuitively to the class of intransitive verbs: *plays*, *runs*, *eats*, *barks*, etc.; it also corresponds to the denotations for longer phrases like *plays chess* and *barks at the neighbor*. As a useful visualization, we can "plot" this function along its domain and range in Figure 1.2. The x axis lists the set of denotations in the world. Recall that each of these is just an arbitrary set of items, although we will label them as denotations of words. The y axis is simply the set of true and false.

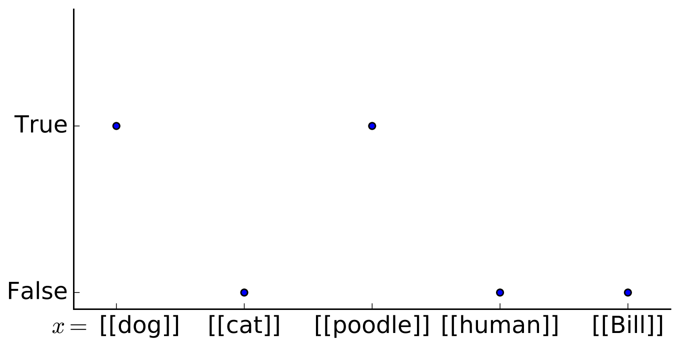


Figure 1.2: A visualization of the denotation of *barks*. The x axis corresponds to denotations of nouns (i.e., sets of entities in the world); the y axis is the domain of truth values.

Importantly, we can keep composing functions from known domains and form new domains corresponding to these new functions. For example, if the domain of intransitive functions defined above is $\mathcal{D}_f$, we can define a class of transitive functions as $f : \mathcal{D}_e \rightarrow \mathcal{D}_f$. We can alternately write this as $f : \mathcal{D}_e \rightarrow (\mathcal{D}_e \rightarrow \mathcal{D}_t)$. The denotation of any span of text is therefore either an entity ($\mathcal{D}_e$), a truth value ($\mathcal{D}_t$), or some class of function inductively defined above.

Notational Note It becomes tedious to write long functions as $\mathcal{D}_e \rightarrow (\mathcal{D}_e \rightarrow \mathcal{D}_t)$; not to mention defining a new set for every function type, as we did for $\mathcal{D}_f$. Therefore, from now on, we'll denote the set of entities as $e = \mathcal{D}_e$, the set of truth values as $t = \mathcal{D}_t$, and functions in the usual way of $a \rightarrow b$. In this notational scheme, intransitive verbs are written as $e \rightarrow t$; transitive verbs are $e \rightarrow (e \rightarrow t)$, and so forth.

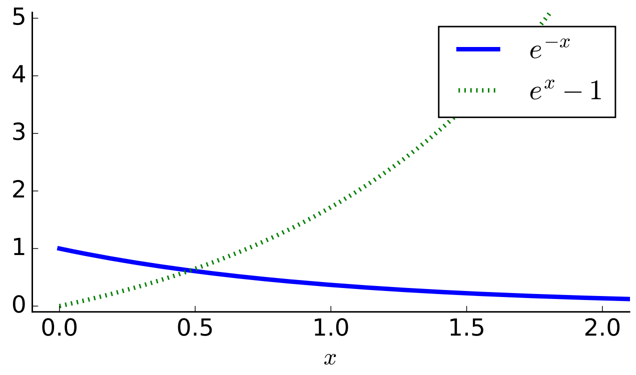
1.1.4 Quantifiers (Operators) are Functions

An important class of lexical items are quantifiers (and more generally natural language operators). From a denotational semantics point of view, these behave just like any other function: *all* has the same denotation as a transitive verb: $e \rightarrow (e \rightarrow t)$; *not* has the same denotation as an intransitive verb: $e \rightarrow t$. However, natural language operators has an important additional property: they are often *monotonic* functions. This property is the happy accident of language that underpins much of the usefulness of monotonicity calculus as a natural logic, and is the main topic of the next section.

1.2 Monotonicity Calculus

We begin this section by reviewing monotonicity in its familiar context: monotonicity of algebraic functions. For example, $f(x) = e^x - 1$ is a *monotone* function – as we increase x, the value of $f(x)$ monotonically increases. A function may also be *antitone*: $f(x) = e^{-x}$ is antitone, since the value of $f(x)$ decreases monotonically as x increases. Lastly, a function can be *nonmonotone* – distinct from antitone – if it is neither monotone nor antitone.

Visually, the plots below show a monotone function ($e^x - 1$) and an antitone function (e^{-x}). We will appeal to analogous visualizations when we move towards working with language.



Monotonicity is an appealing tool because it lets us reason about functions without having to evaluate them. To illustrate, we can define an arbitrarily complex function $f : \mathbb{R} \rightarrow \mathbb{R}$, which we are told is monotone. Without evaluating the function, we are able to conclude that $f(x+1) > f(x)$. This is precisely the type of tool we would like to use to manipulate language: constructing a concrete interpretation of language – like evaluating a complex function – is at best undesirable and at worst impossible. However, if we know some properties about the "monotonicity" of the language, we can manipulate the text such that we preserve some key relations between the original and our mutated text – analogous to the greater-than relation in our algebraic example.

This analogy is much more direct than it may at first appear: we defined a class of functions in Sections 1.1.3 and 1.1.4, and monotonicity calculus will be the calculus of valid inferences that we can draw from reasoning about the monotonicity of these functions. The remainder of this section will explore how to apply monotonicity to the denotational semantics in Section 1.1, and then introduce reasoning about exclusion (Section 1.2.2). The section will conclude by introducing the notion of *polarity*, and exploring how to compose monotone functions in a sentence.

1.2.1 Monotonicity in Language

In generality, a monotone function is a function between partially-ordered sets that preserves the given order. For a function f with domain $\mathbf{X}$ and range $\mathbf{Y}$, we define a partial order over $\leq_X$ and $\leq_Y$. This function is monotone iff:

$$\forall x_1, x_2 \in \mathbf{X} \text{ such that } x_1 \leq_X x_2; \;\; f(x_1) \leq_Y f(x_2) \tag{1.1}$$

We note from Section 1.1 that, by and large, sentences are constructed by composing one or more functions (e.g., verbs, operators). To reason about whether these functions are monotone (or, by extenstion, antitone), we need to show that each of our domains forms a partial order we can define monotonicity against.

First: the domain of noun denotations: $\mathcal{D}_e$ (or, e). We define our partial order $\leq_e$ to be the subset operator: $\subseteq$. That is, if the denotation of a word is completely contained in the denotation of another word, we consider the first word to be "less than" the second. This is intuitively encoding hypernymy as a partial order. For example, $[\![cat]\!] \leq_e [\![feline]\!]$ because any entity which is a cat is also necessarily a feline.

Second: the domain of truth values: $\mathcal{D}_t$ (or, t). Here, we axiomatically define a partial order $\leq_t$ to be:[1]

$$\begin{aligned} false &\leq_t false \\ false &\leq_t true \\ true &\leq_t true \end{aligned}$$

The very important observation to make at this point is that **the partial order $\leq_t$ corresponds exactly to the material conditional $\supset$**. So, for any two propositions A and B, A entails B ($A \supset B$) is the same as $A \leq_t B$. This is the key insight tying together the concepts of monotonicity and entailment. We also point out the unfortunate fact that the symbol for the material conditional is the same as the superset symbol, which would seem to evoke "greater than" rather than the "less than" it actually corresponds to. Unfortunately, this thesis is not the place to redefine either of these symbols.

Lastly: we must define a partial order over our inductively defined function types. A function is less than another function if, for all values in the domain of the functions, the value of the first function is less than the value of the second. Formally: for two functions f and g with the same domain and range $\mathbf{X} \rightarrow \mathbf{Y}$, we say $f \leq_f g$ iff:

$$\forall x \in \mathbf{X};\;\; f(x) \leq_Y g(x) \tag{1.2}$$

For the remainder of this thesis, we will collapse all of these partial orders – $\leq_e$, $\leq_t$, and $\leq_f$ – into a single symbol: $\sqsubseteq$.

Monotonicity is Entailment-Preserving The most important insight from the partial orders above is that our partial order over truth values corresponds exactly to entailment. Although "entailment" is not particularly well-defined for the other denotation types (i.e., does *cat* entail *animal*?), for the purposes of monotonicity calculus and natural logic we

[1]The elegance of the interpretation of false as the empty set and true as a singleton set becomes clear here: we can define the partial order over truth values to be the same subset operator as the partial order over noun denotations.

will take the symbol $\sqsubseteq$ to be entailment.[2] By extension, we can define $A \sqsupseteq B$ to mean that $B \sqsubseteq A$, and define $\equiv$ to be equivalence. That is to say, $A \equiv B$ is the same as $A \sqsubseteq B$ and $B \sqsubseteq A$.

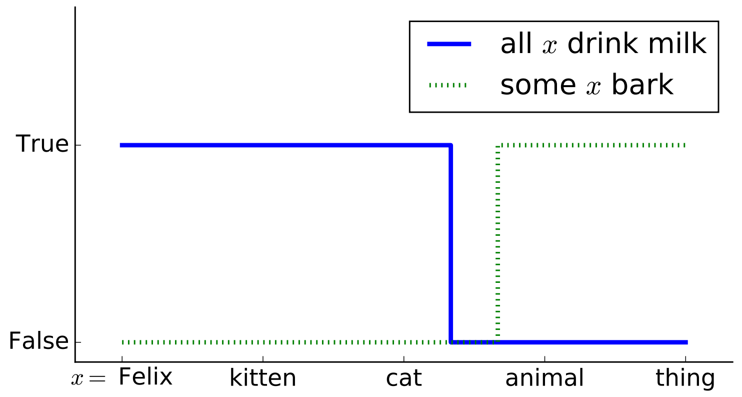
This means that monotone functions are entailment preserving. If a sentence is true, and the function used to construct its denotation (i.e., truth) is monotone with respect to the denotation of a word, then replacing that word with another word whose denotation is a superset of the original word will maintain the truth of the sentence. Taking a concrete example: *all* (a function $e \rightarrow (e \rightarrow t)$) is antitone in its first argument and monotone in its second. So, the sentence *all cats drink milk* is antitone with respect to *cats* and monotone with respect to *drink milk*. Furthermore, we know that $[\![drink\ milk]\!] \sqsubseteq [\![drink\ dairy]\!]$ because $[\![milk]\!] \sqsubseteq [\![dairy]\!]$. Therefore, by the definition of monotonicity, we can replace *drink milk* with *drink dairy* and the resulting sentence (*all cats drink dairy*) is guaranteed to be true if the original sentence was true.

The fact that quantifiers and other operators in natural language have this property of monotonicity is wonderfully convenient. Grounding an interpretation for *all cats drink milk* would be rather difficult in the general case – and certainly difficult for longer utterances. But by appealing to the monotonicity of quantifiers, we do not need to ground the sentence to run an entailment proof on it. Antitone functions behave analogously to monotone functions, but with the direction of the lexical mutation reversed. For example, if *all cats drink milk*, we can infer that *all kittens drink milk* because $[\![cat]\!] \sqsupseteq [\![kitten]\!]$.

Like the visualization with monotone algebraic functions earlier in the section, we can visualize monotonicity over denotations. In the chart below, the x axis is an ordering over denotations.[3] The y axis is the ordering over truth values. We are plotting two functions: *all x drink milk* – antitone in x; and *some x bark* – monotone in x:

[2]This is, again, not to be confused with the symbol for entailment in propositional logic: $\supset$.

[3]In general this is a partial order; however, partial orders are difficult to plot on an axis.



Monotonicity Calculus as a proof theory At this point, we can begin looking at monotonicity calculus as the sort of natural logic proof system we demonstrated at the beginning of the chapter. For example, our inference from *the cat ate a mouse* to *the carnivore ate an animal* could be formally justified with the following proof. We note that the quantifier *the* – like *some* – is monotone in both of its arguments.

1	*The cat ate a mouse.*	
2	*The cat ate a rodent.*	$[\![mouse]\!] \sqsubseteq [\![rodent]\!]$, 1
3	*The cat ate an animal.*	$[\![rodent]\!] \sqsubseteq [\![animal]\!]$, 2
4	*The feline ate an animal.*	$[\![cat]\!] \sqsubseteq [\![feline]\!]$, 3
5	*The carnivore ate an animal.*	$[\![carnivore]\!] \sqsubseteq [\![animal]\!]$, 4

However, we still lack the tools to infer that *no carnivore ate an animal* is false given our premise. For this, we need a theory of how to reason with *exclusion* – which we will review in the next section. Furthermore, our theory currently does not handle nested quantification. In Section 1.2.4 we introduce *Polarity* and the mechanism for propagating monotonicity information to determine the "monotonicity" of a sentence composed of multiple quantifiers.

1.2.2 Exclusion

Although the monotonicity calculus of Sánchez Valencia [15] can already do a range of interesting inferences, it is nonetheless still a very restricted logic. This section reviews work by MacCartney and Manning [12] and Icard and Moss [6] on how natural logic can be extended to handle negation and antonomy by reasoning about *exclusion*. Much of the notation is adapted from Icard and Moss [6].

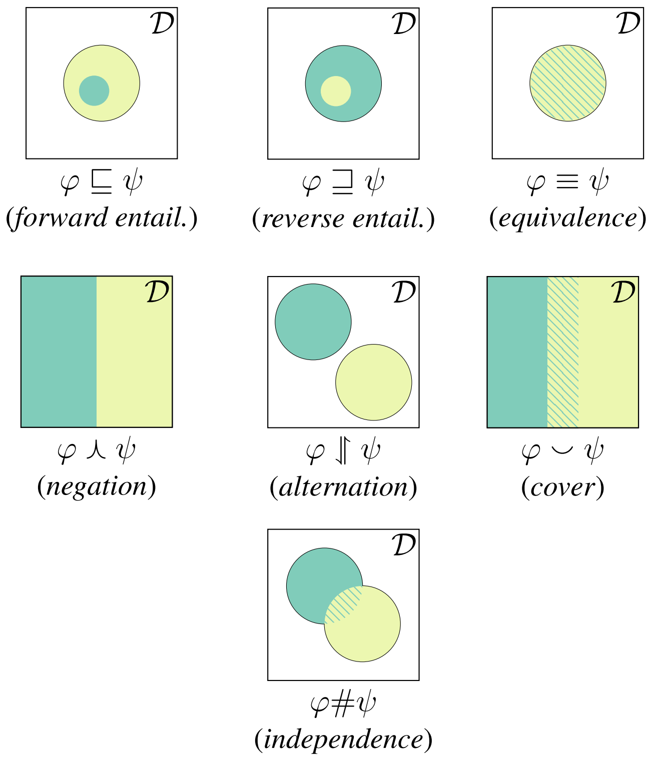


Figure 1.3: An enumeration of the possible relations between two sets φ (dark green) and ψ (light yellow). The top three relations are the simple relations used in monotonicity calculus; the middle three are relevant for exclusion; the bottom relation denotes the case where nothing of interest can be said about the two sets.

The foundations for exclusion come from the observation that there are more relations you can define between sets than the subset / superset / equality relations used in the monotonicity calculus described in Section 1.2.1. Given a set φ and a set ψ, these relations are enumerated in Figure 1.3. The top row correspond to the relations we are familiar with

($\sqsubseteq$, $\sqsupseteq$, $\equiv$) along with their interpretation in natural logic. The middle row describes the three relations relevant for exclusion. The bottom row describes the independence relation, meaning that nothing of interest can be said about the two relations.

We review each of these four new relations, and their interpretation for our three entity types: denotations, truth values, and functions.

First, however; to extend these definition beyond sets, and therefore beyond denotations to truth values and to functions, we introduce a bit of notation. In particular, we define a *partially distributive lattice*. This is a 5-tuple: $(\mathbf{D}, \vee, \wedge, \bot, \top)$, consisting of a domain $\mathbf{D}$ (the set of entities in the lattice), two binary operators $\vee$ and $\wedge$ corresponding to a generalized sense of *maximum* and *minimum* respectively,[4] and two elements in $\mathbf{D}$, $\bot$ and $\top$, corresponding intuitively to the smallest and largest element of $\mathbf{D}$, respectively.

For denotations, we define this lattice as follows, mirroring previous sections.

$\mathbf{D}$ is the power set of all denotations in $\mathcal{D}_e$.
$\vee$ is the union operator: $\cup$.
$\wedge$ is the intersect operator: $\cap$.
$\bot$ is the empty set: $\{\}$.
$\top$ is the full domain of denotations: $\mathcal{D}_e$.

For truth values, we define the lattice straightforwardly as:

$\mathbf{D}$ is the set $\{0, 1\}$, where 0 corresponds to false and 1 corresponds to true.
$\vee$ maximum (i.e., $1 \vee 0 = 1$)
$\wedge$ minimum (i.e., $1 \wedge 0 = 0$)
$\bot$ is false: 0.
$\top$ is true: 1.

Defining this lattice over functions is a bit more verbose, but intuitively analogous to the denotation and truth value cases. Since the range of our functions is ordered (e.g., the domain of truth values t is ordered in a function $e \rightarrow t$), and the range has a maximum and minimum element (e.g., true and false respectively for t), we can from this define our "largest" and "smallest" functions $\top$ and $\bot$. $\top$ is the function which takes any element in

[4]$\vee$ and $\wedge$ must also be commutative, associative, and idempotent; and they must distribute over each other.

its domain, and maps it to the maximum element in the function's range. As a concrete example, for a function $e \rightarrow t$ this corresponds to a tautology – e.g., x *is* x maps everything to $\top$ in the domain of truth values (i.e., *true*). The function corresponding to $\bot$, conversely, maps every element in its domain to the smallest element (i.e., $\bot$) in the function's range.

We further define the $\vee$ and $\wedge$ of two function to be the element-wise $\vee$ and $\wedge$ of the functions. That is, for functions $f : A \rightarrow B$ and $g : A \rightarrow B$, $h = f \vee g$ iff $\forall x, h(x) = f(x) \vee g(x)$. $f \vee g$ is defined analogously. To summarize, the lattice for a function $f : A \rightarrow B$ is:

$\mathbf{D}$ is the set of all functions from A to B.
$\vee$ is the element-wise $\vee$ of the functions.
$\wedge$ is the element-wise $\wedge$ of the functions.
$\bot$ is the function mapping any A to $\bot$ in B.
$\bot$ is the function mapping any A to $\top$ in B.

We can use our definition of this lattice to formally define the new relations in Figure 1.3, and give new interpretations to the relations from above.

Negation ($\curlywedge$) From Figure 1.3, two sets are in *negation* if the union of the sets is the entire domain, and the intersection of the sets is empty. That is, for two sets φ and ϕ, $\varphi \cup \psi = \mathcal{D}$ and $\varphi \cap \psi = \{\}$. Generalizing this to our lattice definition above, we say that two terms are in negation with each other iff $x \vee y = \top$ and $x \wedge y = \bot$. As the name would imply, the most natural examples of negation appearing for pairs of denotations usually involve some sort of morphological negation:

⟦*cat*⟧ $\curlywedge$ ⟦*noncat*⟧
⟦*living thing*⟧ $\curlywedge$ ⟦*nonliving thing*⟧
⟦*possible thing*⟧ $\curlywedge$ ⟦*impossible thing*⟧

For truth values, negation (unsurprisingly) corresponds to logical negation. We can recover the truth table for negation straightforwardly from the definition of the lattice, recalling that $\vee$ is $\max$, $\wedge$ is $\min$, $\top$ is 1, and $\bot$ is 0:

x	y	$x \vee y$	$x \wedge y$	$x \vee y = \top$ and $x \wedge y = \bot$
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

The definition of negation for functions likewise follows. Two functions f and g are then negations of each other ($f \curlywedge g$) iff elementwise $\max(f, g)$ (i.e., $f \vee y$) always maps to $\top$, and $\min(f, g)$ (i.e., $f \wedge y$) always maps to $\bot$. To illustrate with a concrete example, the functions x *is living* and x *is nonliving* are in negation, since for any x it is either true that it is living, or it's true that it is nonliving; and, for any x, it is it is never true that it is both living and nonliving. This extends trivially to quantifiers. For example, *no* and *some* are negations of each other.

The next two relations – alternation ($\downharpoonleft\upharpoonright$) and cover ($\smile$) – can be though of as holding one, but not both of the conditions of negation. In particular, two entities in the negation relation are also necessarily in the alternation and cover relations.

Alternation ($\downharpoonleft\upharpoonright$) Alternation can be thought of as another form of negation, which is weaker in some important respects which will become clear later. In the generalized case, two entities are in alternation iff $x \wedge y = \bot$.

Two denotations are in *alternation* if their intersection is empty. That is, for sets φ and ψ, $\varphi \cap \psi = \{\}$, but unlike negation we do not know anything about their union. This is commonly the relation which holds between antonyms and otherwise contradictory nouns:

$[\![$*cat*$]\!] \downharpoonleft\upharpoonright [\![$*dog*$]\!]$
$[\![$*genius*$]\!] \downharpoonleft\upharpoonright [\![$*idiot*$]\!]$
$[\![$*good deed*$]\!] \downharpoonleft\upharpoonright [\![$*bad deed*$]\!]$

For truth values, alternation equates pragmatically to negation in the context of proving entailment. That is, false $\downharpoonleft\upharpoonright$ false is true (whereas false $\curlywedge$ false is false); however, this is only relevant if we are assuming that our premise is false. Since we are (axiomatically) assuming that our premise is true, this case will never arise, and the truth table looks otherwise equivalent to full negation:

x	y	$x \vee y$	$x \wedge y$	$x \wedge y = \bot$
0	0	0	0	1
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

The intuition for when functions are in alternation becomes potentially hairy, but not awful. Adjective antonyms are a clear example: *hot* x $\rfloor$ *cold* x, since for any x we know that x is not both hot and cold. The quantifiers *all* and *no* are similarly in alternation.

Cover ($\smile$) In many ways, cover is the most strange of the relations in this section. For *nearly* all intents and purposes, this relation indicates neither entailment nor negation between its two entities, but it occasionally conveys a hint of negation. Concretely, cover behaves as negation when reasoning about a counter-factual premise – e.g., if in an entailment chain you have negated your premise – and are now continuing to reason about this presumed false intermediate statement. Formally, two entities are in the cover relation iff $x \vee y = \top$.

For denotations, this is a quintessentially rare case ($\varphi \cup \psi = \mathcal{D}$), and examples which don't amount to outright negation are almost always a bit contrived:

$[\![$*animal*$]\!] \smile [\![$*non-cat*$]\!]$
$[\![$*smartphone*$]\!] \smile [\![$*non-iphone*$]\!]$

The behavior of the cover relation becomes a bit more apparent in the case of truth values. Analogous to how alternation ($\rfloor$) was pragmatically negation when the premise is true, cover is pragmatically negation when the premise is *false*. We will, of course, never assume that the premise in a proof is false; but certainly intermediate steps in the proof may find us with a presumed false statement. In these cases, the cover relation allows us to "negate" this false statement.

x	y	$x \vee y$	$x \wedge y$	$x \vee y = \top$
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

The cover relation is virtually unseen for functions, although of course the definition still carries over fine.

Independence ($\#$) The last relation is independence, which corresponds to no relation holding purely by virtue of the constructed lattice. This is the case for, e.g.,

$[\![$*cat*$]\!] \# [\![$*black animal*$]\!]$
$[\![$*happy*$]\!] \# [\![$*excited*$]\!]$
$[\![$*play*$]\!] \# [\![$*run*$]\!]$

The old relations ($\sqsubseteq$, $\sqsupseteq$, $\equiv$) The relations from the simple monotonicity calculus in Section 1.2.1 of course also have generalized interpretations in the context of our lattice. The behavior remains identical to before. The generalized definitions are:

$x \sqsubseteq y$ iff $x \wedge y = x$
$x \sqsupseteq y$ iff $x \vee y = x$
$x \equiv y$ iff $x \wedge y = x$ and $x \vee y = x$

1.2.3 Proofs with Exclusion

We showed how to run simple proofs in monotonicity calculus in Section 1.2.1 by simply appealing to the transitivity of the $\sqsubseteq$ relation. In effect, we implicitly defined a transitivity table, or *join table*, for how to assess the relation between x and z if we know the relation between x and y, and between y and z. We then *join* these two relations together with the $\bowtie$ operator defined below to obtain the final relation between x and z:

$\bowtie$	$\equiv$	$\sqsubseteq$	$\sqsupseteq$	$\curlywedge$	$\downharpoonleft\upharpoonright$	$\smile$	$\#$
$\equiv$	$\equiv$	$\sqsubseteq$	$\sqsupseteq$	$\curlywedge$	$\downharpoonleft\upharpoonright$	$\smile$	$\#$
$\sqsubseteq$	$\sqsubseteq$	$\sqsubseteq$	$\#$	$\downharpoonleft\upharpoonright$	$\downharpoonleft\upharpoonright$	$\#$	$\#$
$\sqsupseteq$	$\sqsupseteq$	$\#$	$\sqsupseteq$	$\smile$	$\#$	$\smile$	$\#$
$\curlywedge$	$\curlywedge$	$\smile$	$\downharpoonleft\upharpoonright$	$\equiv$	$\sqsupseteq$	$\sqsubseteq$	$\#$
$\downharpoonleft\upharpoonright$	$\downharpoonleft\upharpoonright$	$\#$	$\downharpoonleft\upharpoonright$	$\sqsubseteq$	$\#$	$\sqsubseteq$	$\#$
$\smile$	$\smile$	$\smile$	$\#$	$\sqsupseteq$	$\sqsupseteq$	$\#$	$\#$
$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$	$\#$

Table 1.1: The join table as taken from Icard [5]. Entries in the table are the result of joining a row with a column. Note that the $\#$ always joins to yield $\#$, and $\equiv$ always joins to yield the input relation.

$\bowtie$	$\equiv$	$\sqsubseteq$	$\sqsupseteq$
$\equiv$	$\equiv$	$\sqsubseteq$	$\sqsupseteq$
$\sqsubseteq$	$\sqsubseteq$	$\sqsubseteq$	$\#$
$\sqsupseteq$	$\sqsupseteq$	$\#$	$\sqsupseteq$

As expected, we see the the transitivity of $\sqsubseteq$: this is the key property we needed to run our proofs. We can now define a similar (if larger) join table for our full set of relations. This table is given in Table 1.1.

However, a much more convenient representation of this join table is as a finite state machine. We further will show that we can losslessly collapse this finite state machine into only three intuitive inference states. These observations allow us to formulate a proof as a path through this collapsed state machine, making reasoning with exclusion almost as simple as the original monotonicity proofs.

We construct a finite state machine over states $s \in \{\sqsubseteq, \sqsupseteq, \dots\}$. A machine in state s_i corresponds to relation s_i holding between the initial premise and the derived fact so far. States therefore correspond to states of *logical validity*. The start state is $\equiv$. Outgoing transitions correspond to *inference steps*. Each transition is labeled with a projected relation $\rho(r) \in \{\sqsubseteq, \sqsupseteq, \dots\}$, and spans from a source state s to a target s' according to the join table. That is, the transition $s \xrightarrow{\rho(r)} s'$ exists iff $s' = s \bowtie \rho(r)$. Figure 1.4a shows the automaton, with trivial edges omitted for clarity.

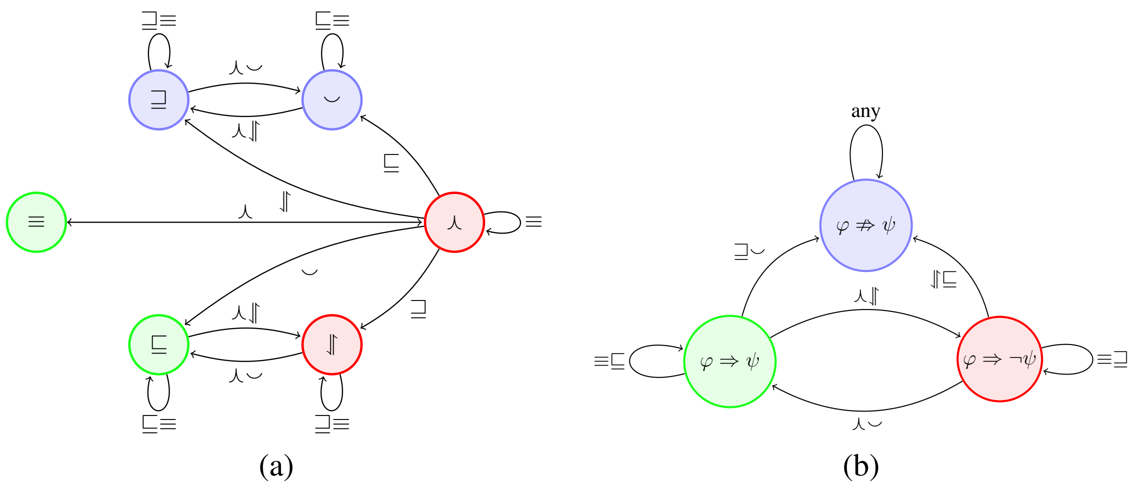


Figure 1.4: (a) Natural logic inference expressed as a finite state automaton. Omitted edges go to the unknown state ($\#$), with the exception of omitted edges from $\equiv$, which go to the state of the edge type. Green states ($\equiv$, $\sqsubseteq$) denote valid inferences; red states ($\downharpoonleft\upharpoonright$, $\curlywedge$) denote invalid inferences; blue states ($\sqsupseteq$, $\smile$) denote inferences of unknown validity. (b) The join table collapsed into the three meaningful states over truth values.

We further collapse this automaton into the three meaningful states we use as output: *valid* ($\varphi \Rightarrow \psi$), *invalid* ($\varphi \Rightarrow \neg\psi$), and *unknown validity* ($\varphi \nRightarrow \psi$). We can cluster states in Figure 1.4a into these three categories. The relations $\equiv$ and $\sqsubseteq$ correspond to valid inferences; $\curlywedge$ and $\downharpoonleft\upharpoonright$ correspond to invalid inferences; $\sqsupseteq$, $\smile$ and $\#$ correspond to unknown validity. This clustering mirrors that used by MacCartney for his textual entailment experiments.

Collapsing the FSA into the form in Figure 1.4b becomes straightforward from observing the regularities in Figure 1.4a. Nodes in the valid cluster transition to invalid nodes always and only on the relations $\curlywedge$ and $\downharpoonleft\upharpoonright$. Symmetrically, invalid nodes transition to valid nodes always and only on $\curlywedge$ and $\smile$. A similar pattern holds for the other transitions.

Formally, for every relation r and nodes a_1 and a_2 in the same cluster, if we have transitions $a_1 \xrightarrow{r} b_1$ and $a_2 \xrightarrow{r} b_2$ then b_1 and b_2 are necessarily in the same cluster. As a concrete example, we can take $r = \curlywedge$ and the two states in the *invalid* cluster: $a_1 = \curlywedge$, $a_2 = \downharpoonleft\upharpoonright$. Although $\curlywedge \xrightarrow{\curlywedge} \equiv$ and $\downharpoonleft\upharpoonright \xrightarrow{\curlywedge} \sqsubseteq$, both $\equiv$ and $\sqsubseteq$ are in the same cluster (*valid*). It is not trivial *a priori* that the join table should have this regularity, and it certainly simplifies the logic for inference tasks.

We can now return to our running example, augmented with negation, and prove that if *the cat ate a mouse* then it is false that *no carnivores ate an animal*. At each inference step, we note the transition we took to reach it form the previous statement, and the new state we are in.

1	*The cat ate a mouse.*		
2	*The cat ate a rodent.*	rel: $\sqsubseteq$	state:$\Rightarrow$, 1
3	*The cat ate an animal.*	rel: $\sqsubseteq$	state:$\Rightarrow$, 2
4	*The feline ate an animal.*	rel: $\sqsubseteq$	state:$\Rightarrow$, 3
5	*The carnivore ate an animal.*	rel: $\sqsubseteq$	state:$\Rightarrow$, 4
6	*No carnivore ate an animal.*	rel: $\curlywedge$	state:$\Rightarrow \neg$, 4

We then notice that the final state we end up in is $\Rightarrow \neg$ – that is, negation. Taking another example, to prove that if Spock is logical, then he is not very illogical:

1	*Spock is logical*		
2	*Spock is illogical*	rel: $\curlywedge$	state:$\Rightarrow \neg$, 1
3	*Spock is very illogical*	rel: $\sqsupseteq$	state:$\Rightarrow \neg$, 2
4	*Spock is not very illogical*	rel: $\curlywedge$	state:$\Rightarrow$, 3

A few final observations deserve passing remark. First, even though the states $\sqsupseteq$ and $\smile$ appear meaningful, in fact there is no "escaping" these states to either a valid or invalid inference. Second, the hierarchy over relations presented in Section 1.2.2 becomes even more apparent – in particular, $\curlywedge$ always behaves as negation, whereas its two "weaker" versions ($|$ and $\smile$) only behave as negation in certain contexts. Lastly, with probabilistic inference, transitioning to the unknown state can be replaced with staying in the current state at a (potentially arbitrarily large) cost to the confidence of validity. This allows us to make use of only two states: *valid* and *invalid*.

1.2.4 Polarity: Composing Monotonicity

So far, we have been talking primarily about natural logic relations between lexical items. That is, $[\![\textit{cat}]\!] \sqsubseteq [\![\textit{animal}]\!]$, and $[\![\textit{black cat}]\!] \sqsubseteq [\![\textit{cat}]\!]$, and $[\![\textit{happy}]\!] \downharpoonleft\!\upharpoonright [\![\textit{sad}]\!]$, and true $\sqsubseteq$ true, and so forth. For our proofs over sentences, we have limited ourselves to a single quantifier, which has allowed us to reason about inferences directly based off of the monotonicity of the quantifier. In this section, we explore two important concepts to complete our theory of monotonicity calculus: we describe how we compose monotonicity when a lexical item is under the scope of multiple quantifiers, and we formally characterize how we can account for apparent nuances between the behaviors of different quantifiers when dealing with exclusion.

To motivate the discussion, we can consider a simple inference:

$$\begin{array}{r|ll} 1 & \textit{No cats don't eat meat} & \\ \hline 2 & \textit{No cats don't eat food} & \quad [\![\textit{meat}]\!] \sqsubseteq [\![\textit{food}]\!], 1 \end{array}$$

Here, *meat* is under the scope of two quantifiers: *no* and *n't* (not).[5] Recall that both of these quantifiers are downward monotone with respect to both of their arguments. For example, when under the scope of a single negation, the following inference is invalid:

$$\begin{array}{r|ll} 1 & \textit{No mice eat meat} & \\ \hline 2 & !\ \textit{No mice eat food} & \quad [\![\textit{meat}]\!] \sqsubseteq [\![\textit{food}]\!], 1 \end{array}$$

To address this, we introduce a notion of *polarity*. Polarity is a property we assign to lexical items. It can be thought of as a function which takes as input a relation which holds between a lexical item and its mutation, and produces as output the relation that holds between the containing sentences. For instance, the polarity of *meat* in the sentence *No mice eat meat* would be a function which translates $\sqsubseteq$to $\sqsupseteq$ and $\sqsupseteq$ to $\sqsubseteq$. The polarity of *meat* in the sentence *No cats don't eat meat* would be a function translating $\sqsubseteq$ to $\sqsubseteq$ and $\sqsupseteq$ to $\sqsupseteq$. This is no different than monotonicity, and is in fact no more than the composition of the monotonicities of the quantifiers acting on a lexical item.

[5]Technically, we are abusing terminology; not all quantifiers in natural logic are quantifiers in the linguistic sense.

The algorithm for determining polarity is simple. Beginning from the lexical item in question, we list the quantifiers which have that lexical item in their scope, and order that list from narrowest to broadest scope. This gives us a list $q_0, q_1, \dots q_n$. In our example, *No cats don't eat meat*, the ordering for *meat* would be $q_0 =$ *n't* and $q_1 =$ *no*. We begin with the identity function as our polarity; then, for each quantifier q_i, we compose our polarity so far with that function's monotonicity.

In the simplest case, this takes the form of flipping an item's polarity between *upward* to *downward* and visa versa for every downward monotone quantifier in its scope. For our double negation case above, we begin with an upward polarity, and flip the polarity twice (once for *no* and once for *n't*) to arrive back at an upward polarity context. As we'll see in the next section, this process becomes more nuanced once we get into the exclusion relations ($\downharpoonleft\!\upharpoonright$, $\curlywedge$, $\smile$), but for now this intuition is enough to formulate a complete proof theory for monotonicity calculus.

Following a variant of the notation in MacCartney and Manning [12], we can a natural logic proof as a table. Each row corresponds to a single lexical mutation on the previous row; the first row is the premise fact. The first column is the (possibly partial) hypothesis. The second column is the lexical relation induced by the mutation performed to obtain the given row. The third column is this relation *projected* up the sentence, based on the lexical item's polarity. The last column is the truth state of the proof, as determined by the previous proof state and the projected relation (see Figure 1.4). To make this concrete, Table 1.2 shows an example inference from *no cats don't eat meat* negating that *black cats don't eat food.*

1.2.5 Additive and Multiplicative Quantifiers

The last topic for this section deals with polarity when we are projecting one of the exclusion relations through the sentence. For example, based on the proof theory in Section 1.2.4, if we assume that the exclusion relations propagate up with the identity relation, we'd get incorrect entailments like the following:

Sentence	**Lexical Rel**	**Projected Rel**	**Truth**
No cats don't eat meat			$\Rightarrow$
No cats don't eat ***food***	$\sqsubseteq$	$\sqsubseteq$	$\Rightarrow$
No ***black cats*** *don't eat food*	$\sqsupseteq$	$\sqsubseteq$	$\Rightarrow$
Some *black cats don't eat food*	$\curlywedge$	$\curlywedge$	$\Rightarrow \neg$

Table 1.2: A tabular proof negating *no carnivores eat animals* from *the cat ate a mouse*. The first column tracks the sentence as it mutates. The second column tracks the lexical natural logic relation induced by the mutation. The third column tracks the lexical relation projected up the sentence. The last column tracks the truth state of the proof, as determined by the FSA in Figure 1.4.

Sentence	**Lexical Rel**	**Projected Rel**	**Truth**
Every cat has a tail			$\Rightarrow$
Every ***dog*** *has a tail*	$\downharpoonleft\!\upharpoonright$	$\downharpoonleft\!\upharpoonright$	$\Rightarrow \neg$

That is to say, despite the fact that $[\![cat]\!] \downharpoonleft\!\upharpoonright [\![dog]\!]$, it's not the case that every cat having a tail contradicts every dog having a tail. To understand why, we introduce the notion of *multiplicative* and *additive* quantifiers, described in more detail in Icard and Moss [6].

Recall that quantifiers are simply functions that map from, e.g., denotations to truth values. Recall further that each of our domains (denotations, truth values, etc.) can be described in terms of a partially distributive lattice: $(\mathbf{D}, \vee, \wedge, \bot, \top)$. The operator $\vee$ is intuitively a union operator, $\wedge$ is intuitively an intersect operator. An upwards monotone quantifier is then:

- *multiplicative* iff $f(x \wedge y) = f(x) \wedge f(y)$.
- *additive* iff $f(x \vee y) = f(x) \vee f(y)$.

Notice that a quantifier can be both additive and multiplicative, and can also be neither additive or multiplicative. Conversely, a downwards monotone quantifier can be anti-additive and anti-multiplicative:

- *anti-multiplicative* iff $f(x \vee y) = f(x) \vee f(y)$.

	$\sqsubseteq$	$\sqsupseteq$	$\curlywedge$	$\downharpoonleft\upharpoonright$	$\smile$
upward	$\sqsubseteq$	$\sqsupseteq$	#	#	#
additive	$\sqsubseteq$	$\sqsupseteq$	$\smile$	#	$\smile$
multiplicative	$\sqsubseteq$	$\sqsupseteq$	$\downharpoonleft\upharpoonright$	$\downharpoonleft\upharpoonright$	#
additive+multiplicative	$\sqsubseteq$	$\sqsupseteq$	$\curlywedge$	$\downharpoonleft\upharpoonright$	$\smile$
downward	$\sqsupseteq$	$\sqsubseteq$	#	#	#
anti-additive	$\sqsupseteq$	$\sqsubseteq$	$\downharpoonleft\upharpoonright$	#	$\downharpoonleft\upharpoonright$
anti-multiplicative	$\sqsupseteq$	$\sqsubseteq$	$\smile$	$\smile$	#
anti-additive+anti-multiplicative	$\sqsupseteq$	$\sqsubseteq$	$\curlywedge$	$\smile$	$\downharpoonleft\upharpoonright$

Table 1.3: The definition of the monotonicity for quantifiers marked with additivity / multiplicativity information. Given that the argument in the domain of the quantifier is mutated according to the relation in the header, the resulting element in the range of the quantifier is guaranteed to have mutated according to the relation in this table. That is, for example, for an additive quantifier $f : \mathcal{X} \rightarrow \mathcal{Y}$, if $x_0 \in \mathcal{X}$, $x_1 \in \mathcal{X}$, $f(x_0) = y_0$, $f(x_1) = y_1$, and $x_0 \curlywedge x_1$, then we can say that $y_0 \smile y_1$.

- *anti-additive* iff $f(x \wedge y) = f(x) \wedge f(y)$.

From our example above, we notice that according to this definition *every* is anti-additive in its first argument: *Every nook and cranny was searched* is equivalent to *every nook was searched* and *every cranny was searched*; but, *every Monday or Wednesday it rains* does implies neither that it rains every Monday, or that it rains every Wednesday.[6] We can also notice that the second argument of *every* is multiplicative: *every cat likes sleeping and eating* entails that *every cat likes sleeping* and *every cat likes eating*. In a similar vein, *no* is anti-additive in both of its arguments; *some* is both additive and multiplicative in both its arguments; and so forth.

Now that we have a more precise notion of monotonicity beyond *upward* and *downward*, we can more accurately characterize the effect quantifiers have on lexical relations as they project up the sentence. We give this refined function in Table 1.3. Note that for the core monotonicity relations ($\sqsubseteq$ and $\sqsupseteq$), the additivity and multiplicativity of the quantifiers is irrelevant, but that these properties have a rather large effect on the other three

[6]Except on the reading of *Monday or Wednesday* as *Monday and Wednesday* – which is a separate, but quite interesting linguistic issue.

meaningful relations. Another interesting observation is that the symmetry between cover ($\smile$) and alternation ($\downharpoonleft\upharpoonright$) that we see in the FSA in Figure 1.4 also appears in the monotonicity function. The difference between additive and anti-additive, multiplicative and anti-multiplicative, etc., is simply replacing all instances of $\sqsubseteq$ with $\sqsupseteq$ (from the definition of monotonicity), and all instances of $\smile$ with $\downharpoonleft\upharpoonright$, and visa versa.

This concludes our exploration of monotonicity calculus. The subsequent section explores some ideas in extending natural logics to propositional reasoning; the remainder of the thesis thereafter will explore applications of natural logic to large-scale open-domain reasoning problems.

1.3 A Propositional Natural Logic

So far all of our natural logic proofs have had a single premise. This is not accidental – a key shortcoming of monotonicity calculus and modern natural logics is that by their very nature they operate on a single premise. That premise is mutated in well-specified ways which preserve entailment; but there's no theory of how to combine multiple premises together for an inference. For example, the disjunctive syllogism is not supported by natural logic:

1	*Either Bill or Ted stole the cookies.*
2	*Ted did not steal the cookies.*
3	*Bill stole the cookies.*

This section sketches a theoretical approach for performing these inferences by taking a hybrid propositional and natural logic. In this way, we can leverage the relative strengths of both formalisms. Propositional logic has rich notions of the common connectives: conjunction, disjunction, etc.; monotonicity calculus is complementary to propositional logic in its handling of quantifiers and lexical entailment. At a high level, we would like a propositional logic to handle inferences between natural language propositions, while deferring to a natural logic to handle the entailments within propositions.

We briefly review propositional logic, and then show how we can perform hybrid natural logic and propositional proofs. Lastly, we explore how we might build a practical system to parse a sentence into propositional components, and the expected challenges therein.

1.3.1 Propositional Logic

In propositional logic, we have propositions (usually denoted by capital letters), and connectives between these propositions. A proposition has a truth value in a given model; propositions are things like "cats have tails" or "the sky is blue". A special proposition denotes contradiction ($\perp$) – an axiomatically false statement. The connectives in propositional logic are a subset of the connectives in first order logic: conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$). Propositional logic does not have predicates (e.g., $P(x)$), and we will not be considering equality and identity (i.e., $P = Q$ or $x = y$). Furthermore, for simplicity, we will treat material implication axiomatically in terms of the core connectives:

$$\neg A \vee B \equiv A \rightarrow B$$

This section is not intended to be a formal introduction to propositional proofs, but rather a review of the *natural deduction* system and Fitch-style proofs we will use in the remainder of this section [3, 4, 7]. In particular, for each of our connectives (and the contradiction symbol), we define a rule for when we can *introduce* that connective, and when we can *eliminate* the connective. In the next section, we will show how we can augment these rules with monotonicity calculus.

Conjunction Rules ($\wedge$) The two rules for conjunction are quite straightforward. We can always take as true either side of a conjunction, and we can introduce a conjunction only if we can prove both conjuncts:

$$\begin{array}{r|ll} 1 & A \land B & \\ \hline 2 & A & \land\,\text{E}, 1 \end{array} \qquad \begin{array}{r|ll} 1 & A & \\ 2 & B & \\ \hline 3 & A \land B & \land\,\text{I}, 1, 2 \end{array}$$

$$\begin{array}{r|ll} 1 & A \land B & \\ \hline 2 & B & \land\,\text{E}, 1 \end{array}$$

$\land$ **elimination** $\qquad$ $\land$ **introduction**

Disjunction Rules ($\lor$) Conjunction is, of course, quite boring on its own. Disjunction is a measure more interesting. Introducing disjunction is straightforward: if we know A, we also know that either A or B. However, eliminating a disjunction requires reasoning by cases. We have to show that the same formula is true if it is true for all the disjuncts. Formally, the rules are:

$$\begin{array}{r|lll} 1 & A \lor B & & \\ \hline 2 & \quad & \begin{array}{|l} A \\ \hline \vdots \\ Q \end{array} & \\ \vdots & & & \\ n & & & \\ n+1 & & \begin{array}{|l} B \\ \hline \vdots \\ Q \end{array} & \\ \vdots & & & \\ m & & & \\ m+1 & Q & & \lor\,\text{E}, n, m \end{array} \qquad \begin{array}{r|ll} 1 & A & \\ \hline 2 & A \lor B & \lor\,\text{I}, 1 \end{array}$$

$\lor$ **elimination** $\qquad$ $\lor$ **introduction**

Negation Rules ($\neg$) Negation elimination is straightforward: we can eliminate double negations. Negation can be introduced, in turn, via a proof by contradiction:

$$\begin{array}{r|ll} 1 & \neg\neg A & \\ \hline 2 & A & \neg\text{ E, }1 \end{array} \qquad \begin{array}{r|l|ll} 1 & & A & \\ \cline{3-3} \vdots & & \vdots & \\ n & & \bot & \\ n+1 & \neg A & & \neg\text{ I, }1\text{–}n \end{array}$$

¬ elimination **¬ introduction**

Contradiction Rules (⊥) Contradiction can be introduced when we derive a proposition and its negation. Contradiction elimination is clearly the least intuitive of the deduction rules: if we have derived a contradiction, we can state any proposition as true. The Fitch rules for contradiction are:

$$\begin{array}{r|ll} 1 & \bot & \\ \hline 2 & Q & \bot\text{ E, }1 \end{array} \qquad \begin{array}{r|ll} 1 & A & \\ \hline \vdots & \vdots & \\ n & \neg A & \\ n+1 & \bot & \bot\text{ I, }1, n \end{array}$$

¬ elimination **¬ introduction**

The next section shows how we can adapt these rules to the incorporate natural logic.

1.3.2 A Hybrid Logic

In the previous section, we treated propositions as atomic units. A formula $A \vee B$ has no substructure beyond being a disjunction of two propositions. However, in most cases, these propositions *do* in fact have structure. Furthermore, most of the time we can express a proposition as a sentence. In this section we propose an extension to propositional logic which uses monotonicity calculus to augment the semantics of these atomic propositions (conversely: an extension to monotonicity calculus which incorporates propositional reasoning).

Recall that in natural logic, a sentence is represented as a truth value. Whereas $[\![\textit{cat}]\!]$ is a set of entities which are cats, and verbs like $[\![\textit{run}]\!]$ are denoted as functions from sets to truth values, a complete sentence (e.g., $[\![\textit{cats run}]\!]$) is either *true* or *false*. From this point, the propositional logic side of our proof theory no longer has to care about the substructure of the sentence, and can treat it as an atomic proposition.

Notationally, we define the space of well formed formulas in our hybrid logic analogously to how a formula is defined in propositional logic:

1. $[\![x]\!]$ is a well-formed formula iff $[\![x]\!] \in \{true, false\}$. $[\![\textit{cat}]\!]$ is not a well-formed formula, but $[\![\textit{cats have tails}]\!]$ is.

2. $\perp$ is a well-formed formula.

3. If F is a well-formed formula, so is $\neg F$.

4. If F and G are well-formed formulas, so is $F \wedge G$.

5. If F and G are well-formed formulas, so is $F \vee G$.

We now turn to our hybrid proof theory. Trivially, all of the inference rules of propositional logic hold, if we do not mutate any of the atomic sentences. Likewise, trivially all of the natural logic inferences hold, if they are not embedded in a larger propositional formula. Our task is then twofold: (1) what natural logic inferences are warranted inside of a given propositional formula; and (2) what *additional* propositional inferences can be made by appealing to the semantics of natural logic.

The first of these questions is answered by noticing that the propositional connectives, like natural language quantifiers, have monotonicity. Conjunction ($\wedge$) is upward monotone, and both additive and multiplicative; disjunction ($\vee$) is upward monotone, but only additive. Conjunction is multiplicative because $(A \wedge B) \wedge C \vDash (A \wedge C) \wedge (B \wedge C)$, and additive because $(A \vee B) \wedge C \vDash (A \wedge C) \vee (B \wedge C)$. Analogously, disjunction is not multiplicative: $(A \wedge B) \vee C \nvDash (A \wedge C) \vee (B \wedge C)$; but it is additive: $(A \vee B) \vee C \vDash (A \vee C) \vee (B \vee C)$. Negation, in turn, is downward monotone, and both anti-multiplicative but not anti-additive: $\neg(A \vee B) \vDash \neg A \vee \neg B$, but $\neg(A \wedge B) \nvDash \neg A \wedge \neg B$.

From here, we can use our normal projection rules to mutate lexical items not only in a sentence, but also embedded in a propositional formula. For example, the following is now a valid inference using only natural logic:

$$
\begin{array}{r|ll}
1 & \neg\llbracket \textit{all cats are friendly} \rrbracket \wedge \llbracket \textit{all cats are cute} \rrbracket & \\
\hline
2 & \neg\llbracket \textit{all } \textbf{\textit{felines}} \textit{ are friendly} \rrbracket \wedge \llbracket \textit{all cats are cute} \rrbracket & \llbracket \textit{cat} \rrbracket \sqsubseteq \llbracket \textit{feline} \rrbracket, 1 \\
3 & \neg\llbracket \textit{all felines are friendly} \rrbracket \wedge \llbracket \textit{all } \textbf{\textit{tabby cats}} \textit{ are cute} \rrbracket & \llbracket \textit{cat} \rrbracket \sqsupseteq \llbracket \textit{tabby cat} \rrbracket, 2
\end{array}
$$

Of course, we should be able to now apply the propositional rules from Section 1.3.1:

$$
\begin{array}{r|ll}
4 & \llbracket \textit{all tabby cats are cute} \rrbracket & \wedge\text{ E}, 3
\end{array}
$$

Perhaps the most interesting question is, what *additional* inferences can we draw from this hybrid logic? In particular, our two logics each have their own notion of negation: in monotonicity calculus, $\curlywedge$, $\|$, and $\smile$ each behave a bit like negation. In fact, this lets us define a new negation introduction rule in propositional logic. In short, if we derive a new sentence from a presupposed true premise, and it is in the natural logic relations $\curlywedge$ or $\|$ with the original sentence, we can introduce the negation of the original sentence. Analogously, if we derive a new sentence from a presupposed false premise, and it is in the natural logic relations $\curlywedge$ or $\smile$ with the original sentence, we can introduce the negation of the original sentence (to yield a double negation). This falls directly out of the state formulation of the finite state automata described in Figure 1.4. Formally:

$$\begin{array}{l|ll}
1 & \llbracket X \rrbracket & \text{state:}\Rightarrow \\ \cline{2-2}
\vdots & \vdots & \\
n & \llbracket Y \rrbracket & \text{state:}\neg\Rightarrow,\ 1\text{–}n \\
n+1 & \neg\llbracket X \rrbracket & \neg\ \text{I},\ 1,\ n
\end{array}$$

natural ¬ introduction 1

$$\begin{array}{l|l|ll}
1 & \multicolumn{2}{l}{\neg\llbracket X \rrbracket} & \text{state:}\Rightarrow \\ \cline{2-3}
2 & & \llbracket X \rrbracket & \text{state:}\neg\Rightarrow,\ 1 \\ \cline{3-3}
\vdots & & \vdots & \\
n & & \llbracket Y \rrbracket & \text{state:}\Rightarrow,\ 2\text{–}n \\
n+1 & \multicolumn{2}{l}{\neg\neg\llbracket X \rrbracket} & \neg\ \text{I},\ 1,\ 2\text{–}n
\end{array}$$

natural ¬ introduction 2

The added complication here is that we now have to keep track of whether introduced formulas are presupposed to be true or false. For instance, line 2 in the right ¬ introduction proof presupposes that the sentence $\llbracket X \rrbracket$ is false, and therefore the cover ($\smile$) relation can flip the truth of the sentence to be true. Similarly, in the left proof we assume that our premises are true – this is not necessarily the case if, e.g., you are trying to prove your premises false.[7] If this information is not available *a priori* in the proof, the new negation introduction rules are only warranted by the full negation relation ($\curlywedge$).

We can combine these insights to solve a syntactically simpler version of our motivating example in Figure 1.5.

1.3.3 Shallow Semantic Parsing

The hybrid proof system from Section 1.3.2 lays out an elegant theory for entailment when we are composing atomic sentences. However, in natural language, propositional statements are usually not said in a such a straightforward way. Even taking our simple running example, it's much more natural to say *Either Bill or Ted stole the cookies* than *Bill stole the cookies or Ted stole the cookies*. Therefore, it is clearly useful to have a semantic parser which takes as input a complex utterance, and produces as output a hybrid natural/propositional logic formula. Such a parser would allow us to do powerful forms of logical reasoning, without requiring a heavyweight logical representation (e.g., the Abstract Meaning Representation [1]). In this section, we outline some of the challenges for

[7]This is a problem even in vanilla natural logic proofs. If you believe the premise to be false, you have to start in the false state of the finite state automata in Figure 1.4.

```
1  │ ⟦Bill stole the cookies⟧ ∨ ⟦Ted stole the cookies⟧
2  │ ⟦Ted did not steal the cookies⟧
   ├───────────────────────────────
3  │   │ ⟦Bill stole the cookies⟧
   │   ├────────────────────────
4  │   │ ⟦Bill stole the cookies⟧           R, 3
5  │   │ ⟦Ted stole the cookies⟧
   │   ├────────────────────────
6  │   │ ⟦Ted did steal the cookies⟧        ⟦steal⟧ ≡ ⟦did steal⟧; state:⇒, 5
7  │   │ ⟦Ted did not steal the cookies⟧    ⟦did⟧ ⅄ ⟦did not⟧; state:¬ ⇒, 6
8  │   │ ¬⟦Ted stole the cookies⟧           ¬ I, 5, 7
9  │   │ ⊥                                  ⊥ I, 5, 8
10 │   │ ⟦Bill stole the cookies⟧           ⊥ E, 9
11 │ ⟦Bill stole the cookies⟧               ∨ E, 1, 3–5, 5–10
```

Figure 1.5: A hybrid propositional + natural logic proof showing the disjunctive syllogism.

such a parser.

***And* is not always conjunction** The classic case of this is a sentence like *Jack and Jill are friends*, where of course this sentence should not be parsed as *Jack is friends* and *Jill is friends*. However, there are more subtle cases of this as well. For example, consider the exchange:

> *What would you like for dinner?*
> *Taco Bell and Pizza Hut are my choice.*

Here, the utterance *Taco Bell and Pizza Hut are my choice* has a meaning more akin to *Taco Bell is my choice* or *Pizza Hut is my choice*.

***Or* is not always disjunction** Conversely, *or* does not always imply a disjunction. For example, consider the exchange:

> *Can I have pets in the apartment?*
> *Cats or dogs are ok.*

Clearly here, the semantics of the second statement is: $[\![\textit{cats are ok}]\!] \wedge [\![\textit{dogs are ok}]\!]$.

***Or* is often exclusive** In propositional logic, disjunction is not exclusive: $A \vee B$ does not forbid $A \wedge B$. This is played out in many uses of the word *or*; for example: *you should do your taxes or your homework*. However, in a surprising number of cases, *or* implies an exclusive or. Take, for instance:

> *Do or do not; there is no try.*
> *Either cats or dogs chase mice.*
> *Jack or Jill is responsible for that.*

Handling these and other phenomena in language is a nontrivial problem. However, this is true for any semantic parsing task; and if a good parser can be built then propositional and natural logic provide a fast and expressive paradigm for solving entailment tasks.

The rest of this thesis will use various aspects of natural logic in practical applications, focusing on large-scale textual inference and question answering tasks. The first challenge, addressed by the next chapter, is on extending the proof theory described here to operate over not a single premise, but a very large set of candidate premises. That is, we are using natural logic to look for any supporting premise for a hypothesis in a very large collection of plain text.

Bibliography

[1] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. *Proc. Linguistic Annotation Workshop*, 2013.

[2] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of EMNLP*, 2013.

[3] Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische zeitschrift*, 39(1):176–210, 1935.

[4] Gerhard Gentzen. Untersuchungen über das logische schließen. ii. *Mathematische Zeitschrift*, 39(1):405–431, 1935.

[5] Thomas Icard, III. Inclusion and exclusion in natural language. *Studia Logica*, 2012.

[6] Thomas Icard, III and Lawrence Moss. Recent progress on monotonicity. *Linguistic Issues in Language Technology*, 2014.

[7] Stanisław Jaśkowski. *On the rules of suppositions in formal logic*. Nakładem Seminarjum Filozoficznego Wydziału Matematyczno-Przyrodniczego Uniwersytetu Warszawskiego, 1934.

[8] Hans Kamp. Two theories about adjectives. 1975.

[9] Rohit J. Kate. Transforming meaning representation grammars to improve semantic parsing. In *CoNLL*, Manchester, UK, 2008.

[10] P. Liang, M. I. Jordan, and D. Klein. Learning dependency-based compositional semantics. In *ACL*, 2011.

[11] Percy Liang and Christopher Potts. Bringing machine learning and compositional semantics together. *Annual Review of Linguistics*, 1(1):355–376, 2015.

[12] Bill MacCartney and Christopher D Manning. Modeling semantic containment and exclusion in natural language inference. In *Coling*, 2008.

[13] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *VLDB*, 2011.

[14] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

[15] Víctor Manuel Sánchez Sánchez Valencia. *Studies on natural logic and categorial grammar*. PhD thesis, University of Amsterdam, 1991.

[16] Johan van Benthem. *Essays in logical semantics*. Springer, 1986.

[17] Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*. AUAI Press, 2005.

[18] Ce Zhang and Christopher Ré. Dimmwitted: A study of main-memory statistical analytics. *VLDB*, 2014.