Introduction to the Mathematics of Language

Michael Hammond
U. of Arizona

August 7, 2007
## Contents

1 Overview .......................... 5  
1.1 A Scientific Approach ................. 5  
1.2 Other Reasons for Formal Theories .......... 7  
1.3 An example ............................ 8  
1.4 Organization of the Book .................. 8  
1.5 Acknowledgements ....................... 9  

2 Language .......................... 10  
2.1 Common Misconceptions ................. 10  
2.1.1 Learning is memorization ................. 10  
2.1.2 Correction is necessary to learn language .......... 11  
2.1.3 Some people have bad grammar ............. 12  
2.1.4 Two negatives make a positive ............. 13  
2.1.5 I don’t have an accent .................. 14  
2.1.6 Some languages are logical ................. 15  
2.1.7 Some languages are primitive ............... 15  
2.2 Key Notions .......................... 16  
2.2.1 What is Language? ..................... 17  
2.2.2 Creativity ............................ 18  
2.2.3 Prescriptive vs. Descriptive Grammar ........ 19  
2.2.4 Competence and Performance ............... 19  
2.3 General Areas ......................... 21  
2.3.1 Structures of Language ................. 21  
2.3.2 Other Areas .......................... 28  
2.4 Summary ............................. 29  
2.5 Exercises ............................. 30
## 3 Set Theory

3.1 Sets ................................................. 31
3.2 Membership ........................................ 34
3.3 Operations ......................................... 35
3.4 Fundamental Set-theoretic Equalities ............ 36
3.5 Theorems .......................................... 38
3.6 Ordered Pairs, Relations, and Functions ......... 39
3.7 Language examples ................................ 41
  3.7.1 Examples of Sets ................................ 41
  3.7.2 Examples of Relations and Functions ....... 42
3.8 Summary ............................................ 44
3.9 Exercises ........................................... 45

## 4 Sentential Logic

4.1 The intuition ........................................ 46
4.2 Basic Syntax ........................................ 46
  4.2.1 Simplicity .................................... 48
  4.2.2 Lack of Ambiguity ............................ 49
  4.2.3 Artificiality ................................ 49
4.3 Basic Semantics .................................... 49
4.4 The Meanings of the Connectives ................. 53
  4.4.1 Negation ...................................... 54
  4.4.2 Conjunction .................................. 54
  4.4.3 Disjunction .................................. 55
  4.4.4 Conditional .................................. 55
  4.4.5 Biconditional ................................ 55
4.5 How Many Connectives? ............................ 56
4.6 Tautology, Contradiction, Contingency ........... 58
4.7 Proof ............................................... 59
4.8 Rules of Inference .................................. 68
4.9 Conditional Proof .................................. 71
4.10 Indirect Proof ..................................... 73
4.11 Language ........................................... 74
4.12 Summary ........................................... 75
4.13 Exercises ........................................... 76
5 Predicate Logic 77
  5.1 Syntax ................................................. 77
  5.2 Semantics .............................................. 79
  5.3 Laws and Rules ......................................... 84
    5.3.1 Laws ............................................. 84
    5.3.2 Rules of Inference ................................. 88
  5.4 Proofs ................................................... 89
    5.4.1 Indirect Proof ..................................... 91
    5.4.2 Conditional Proof ................................ 92
  5.5 Summary ................................................ 92
  5.6 Exercises ............................................. 93

6 Formal Language Theory 96
  6.1 Languages .............................................. 96
  6.2 Grammars .............................................. 97
  6.3 Finite State Automata ................................ 105
  6.4 Regular Languages ..................................... 112
  6.5 Automata and Regular Languages ..................... 113
  6.6 Right-linear Grammars and Automata .................. 115
  6.7 Closure Properties of Regular Languages ............. 115
  6.8 Pumping Lemma for Regular Languages ............... 117
  6.9 Relevance of Regular Languages ..................... 119
  6.10 Summary ............................................... 121
  6.11 Exercises ............................................. 122

7 Formal Language Theory Continued 124
  7.1 Context-free Languages .............................. 124
  7.2 Pushdown Automata ................................... 126
  7.3 Equivalence of Non-deterministic PDAs and CFGs .... 129
    7.3.1 CFG to PDA ...................................... 130
    7.3.2 PDA to CFG ...................................... 132
  7.4 Closure Properties of Context-free Languages ...... 135
  7.5 Pumping Lemma for Context-free Languages ........ 136
  7.6 Natural Language Syntax .............................. 138
  7.7 Determinism .......................................... 139
  7.8 Other Machines ....................................... 140
  7.9 Summary ................................................ 144
  7.10 Exercises ............................................. 145
## CONTENTS

8 Probability

8.1 What is it and why should we care? ......................................................................... 147
8.2 Basic Probability ..................................................................................................... 149
8.3 Combinatorics ......................................................................................................... 153
8.4 Laws ....................................................................................................................... 154
8.5 A Tricky Example .................................................................................................... 156
8.6 Conditional Probability ........................................................................................... 157
8.7 Distributions ........................................................................................................... 160
8.8 Summary ................................................................................................................ 165
8.9 Exercises ................................................................................................................. 165

9 Probabilistic Language Models

9.1 The Chain Rule ....................................................................................................... 167
9.2 N-gram models ....................................................................................................... 168
  9.2.1 Unigrams .......................................................................................................... 168
  9.2.2 Bigrams ............................................................................................................ 170
  9.2.3 Higher-order N-grams ...................................................................................... 175
  9.2.4 N-gram approximation ..................................................................................... 175
9.3 Hidden Markov Models ............................................................................................ 177
  9.3.1 Markov Chains ................................................................................................. 178
  9.3.2 Hidden Markov Models .................................................................................... 179
  9.3.3 Formal HMM properties ................................................................................. 180
  9.3.4 Bigrams and HMMs ......................................................................................... 181
  9.3.5 Higher-order N-grams ...................................................................................... 182
9.4 Probabilistic Context-free Grammars ...................................................................... 183
9.5 Summary ................................................................................................................ 189
9.6 Exercises ................................................................................................................. 189
Chapter 1

Overview

The investigation of language can be carried out in many different ways. For example, we might learn to speak a language as a way to understand it. We might also study literature and poetry to the same end. We might even write novels and poetry to understand the intricacies of expression that some language provides.

These are valuable pursuits, not to be denigrated in any way, but they do not provide for a scientific understanding of language, one where we can make falsifiable or testable claims about our object of study. A falsifiable claim is one that can be disproven with real data. For example, if we foolishly hypothesize that Shakespeare wrote good poetry because he wrote in English, we would need some objective way to assess how “good” some piece of poetry is independent of the language the author wrote in. If the hypothesis rested instead entirely on our own ideas about how good individual poems are, then it would surely not be falsifiable, and thus not be science.

1.1 A Scientific Approach

There are, of course, a number of ways to do science with language. We might investigate the range of sentence types that some speaker can produce or that some corpus contains. For example, do male or female characters in Shakespeare have longer sentences? We might look at the set of sounds that can occur in any one language, or the set of sounds used in some specific poetic context. Do the languages of Europe have more vowels than the languages of India? We might want to investigate the time course of language
acquisition or of language change. Do children learn all the vowels of English before or after they learn all the consonants? Are vowels or consonants more “stable” over the history of the English language?

As we lay out these different questions, we beg the question of why we might expect different outcomes. Why do we believe that language should work in any particular way? Our expectations about how language should behave are our theory of language. For example, if we hypothesize, for example, that all languages have the vowel [a], as in English father, then we have an implicit theory about how language works and about the vowel [a]. If we believe that male or female characters have longer sentences in Shakespeare’s plays, then we have a theory of sentence length and its relationship to gender.

These expectations can be fairly informal and intuitive. For example, we might believe that [a] is a very easy vowel to produce and that languages make use of easier sounds before making use of harder sounds. With respect to sentence length, we might have some idea about the role female characters played in Shakespeare and how that would be reflected in the language of those characters.

As we proceed along these lines, as we make, test, and refine our hypotheses about language, it behoves us to make our theory of language more explicit. The more explicit our theory of language is, the more falsifiable it is. As scientists, we want our theory of language to be as testable as possible.

As our theory becomes more explicit, it tends to become more formal. A formal theory is one where the components of the theory are cast in a restricted metalanguage, e.g. math or logic. The languages of math and logic are quite precise. A theory cast in those terms can make very specific predictions that are quite falsifiable. A theory cast in normal English can be quite vague. An analogy would be the language of contracts. Typically, such documents are quite hard to make out for laymen as they are written using very specific language that lawyers have very specific interpretations for. While this kind of language can be quite frustrating for the rest of us, it is essential. The legal metalanguage of contracts provides a mechanism whereby our rights and commitments can be negotiated clearly in contracts.

Likewise, mathematical and logical formalisms can appear daunting as parts of a theory of language, but they can be indispensable to making a theory maximally falsifiable.

Understanding the formalisms used in theories of language requires specialized knowledge of various formal domains. That is the point of this book.
1.2 Other Reasons for Formal Theories

Intuitively, formalism is all the symbols, all the stuff that looks like math. As we’ve discussed in the previous section, the formalism is just a mechanism for being maximally explicit. If we want to build theories that we can test in precise ways, then we need to have theories that are as precise as possible. A proper formalization enables us to do this.

Formalization also enables us to implement our theories computationally. We might want to write a computer program that mimics some aspect of the grammar of a language, its sentences, words, or sounds. For example, imagine we wanted to write a program that would produce poetry. We would supply a vocabulary and the program would spit back a random poem. This may seem rather silly, but is, in fact, a hugely complicated task. We would have to provide the program with some definition of what constitutes poetry. To the extent that this definition had any content, the task becomes quite complex. For example, how do we define “rhyme”? How do we get the program to produce grammatical sentences and not just random strings of words? How do we get the program to figure out how words are actually pronounced? Computer programs are merciless in requiring specific answers to questions like these and an inexplicit theory of language will be of little help.

Why would we want to write such programs? There are two broad reasons. The first is that it is another way to test our understanding of our theory of language. For example, if we have the wrong theory of rhyme, then our program would produce bad poetry. The second is that we may actually want to do something useful in the outside world with our theories, use them for some purpose other than the pursuit of “truth”. While a program that wrote poetry would seem of little use, a program that examined text and identified it as poetry or not might be of great use. In either case, being as explicit as possible in our formalization of our theory makes implementing that theory as painless as possible.

Another reason why we formalize theories of language is to understand the general character of formalization better. For example, if we challenge ourselves to characterize some aspect of language in terms of first-order logic\(^1\), we may find out something about logic too. Thus formalization of theories of language can also tell us about math and logic more generally.

\(^1\)More on this in chapters 4 and 5 below.
1.3 An example

Let’s consider a simple example. Pretty much anyone would agree that “two times two equals four” is the same thing as $2 \times 2 = 4$. In a relatively simple domain like multiplication, there does not appear to be much to be gained by translating words into an equation. Consider on the other hand a sentence like the following:

(1.1) Two women saw two men.

If we are interested in the meanings of sentences, then a bit of formalism might be helpful in characterizing what such a sentence can mean. In the case at hand, this sentence has at least two different interpretations. First, the sentence could mean that there are precisely two women and precisely two men and each of the two men were seen by at least one of the women. However, there is another possible interpretation where there are only two women, but up to four men. Each woman saw two men, but they may not have been the same men. We can make this a little clearer if we assign names to the individuals: Mary, Molly, Mark, Mike, Mitch, and Matt.

(1.2) First interpretation: Mary and Molly saw Mark and Mike.

(1.3) Second interpretation: Mary saw Mark and Mike, and Molly saw Mitch and Matt.

The distinction is a subtle one and can get quite intricate with more complex sentences. Thus characterizing the meanings of sentences might require some fairly elaborate formal machinery.

1.4 Organization of the Book

The main topics we cover in this book are the following.

**Set Theory** Abstract theory of elements and groups. Underlines virtually everything else we’ll treat.

**Logic** A formal system with which we can argue and reason. There are two ways to look at this. One way to look at it is as a formal system
that underlies how we make arguments. However, another way to look at it is as a “perfect language” with syntax and semantics rigorously defined.

**Formal Language Theory** An explicit way to look at languages and the complexity of their description with an eye to how we might compute things about those languages.

**Probability** How to be explicit about likelihood, essential for experimental disciplines, linguistic behavior, language change, sociolinguistics, . . . .

### 1.5 Acknowledgements

Thanks to Andrew Carnie, Chris Cronenberg, Heidi Harley, Shiloh Mossie, and Diane Ohala for useful feedback and discussion. Thanks also to the students in Linguistics 178 and 501 at the University of Arizona. All errors are, of course, my own.
Chapter 2

Language

In this chapter, we outline how language works in very general terms. We begin with a tour of some of the more common and egregious misunderstandings about language and then progress to the key notions of any theory of language. Finally, we briefly review the principal areas of language study.

2.1 Common Misconceptions

Language is a wonderful domain to do science with. All humans have language in some sense, so the data one might need to test one’s hypotheses are ready to hand. Unfortunately, there is a great deal of misunderstanding about how language works. We need to break through these misunderstandings before we can appreciate the formal structures of language.

2.1.1 Learning is memorization

Let’s begin with a fairly common notion: children learn language by memorizing words and sentences. The basic idea here is that all there is to learning a language is memorizing the words and sentences one hears.

There are several problems with this view of language learning. First, it is not altogether clear what is intended by memorization, but the most obvious interpretation of this term would have to be incorrect. Memorization implies a conscious active effort to retain something in memory, but language learning seems to happen in a rather passive fashion. Children do not appear to spend any effort in acquiring the languages they are exposed to, yet they
CHAPTER 2. LANGUAGE

will learn any number of them simultaneously.

Memorization is certainly a tool by which things can be learned, but it is much more appropriate as a way of characterizing how we learn, say, the multiplication tables, rather than how we learn language.

Another problem with the memorization story is that children produce things that they haven’t been exposed to. For example, it is quite typical that at an early stage of acquisition children will produce forms like *foots* for the adult form *feet*, or *goed* for adult *went*, etc. This has no explanation on the memorization view. The child has presumably only heard the correct adult forms *feet*, *went*, etc., yet produces forms that they’ve never been exposed to. Presumably, in the case of *foots*, the child is attempting to produce the plural form by the general rule of adding an *–s*, despite having heard the irregular plural form *feet*. Likewise, in the case of *goed*, the child is attempting to produce the past tense by means of the general rule that says to add *–ed*.

Similar examples can be constructed at the sentence level. Imagine the child has been exposed to a novel situation, say, a new toy or a new person in their life. The child isn’t stumped by these sorts of situations, but constructs sentences appropriate to the new individuals. That is, the child has no trouble saying that *Ernie is in the kitchen*, even if the child has just met Ernie and it’s the first time he’s been in the kitchen.

Both sorts of example suggest that learning is at least partially a process of *generalizing* from what the child has been exposed to, not simple rote memorization of that exposure.

2.1.2 Correction is necessary to learn language

Another common misconception related to language learning is that children need to be corrected by adults to learn their language properly. We’ve already cited examples where children produce forms that are incorrect in the adult language, e.g. *foots*. It is reasonable—though incorrect—to suppose that they need explicit correction to produce the correct form.

There are two glaring problems with this view. First, not all parents correct their children, yet these sorts of errors disappear in the adult language, typically by age three or four.

Another perhaps rather surprising fact is that children typically ignore any attempt at correction. The literature is rife with anecdotes of children being painstakingly corrected for such forms and then blithely continuing to use them.
The point is that these sorts of “errors” are a natural part of language acquisition. They develop and fall away of their own accord, and parental correction plays essentially no role in this.

2.1.3 Some people have bad grammar

Most of us believe that some people have “bad grammar”. This is a very tricky notion so it will be useful to go through some examples. Consider the following pair of sentences.

(2.1) You and me are studying logic.
    You and I are studying logic.

The two sentences mean the same thing. Both are used in English. Some English speakers prefer to use one over the other and some speakers will use both in different contexts. If a speaker does use both, then the difference is one of register or modality. The first is more typical of speech, rather than writing. The first is also more informal than the second.

Is either form better than the other? No. It is certainly the case that they are contextually restricted, but there is no objective scientific reason to maintain that either structure is “better” than the other.

One might note that the second sentence type is older while the first type is newer. This is indeed correct, but it does not make either structure better or worse than the other. In fact, there are cases that work the other way. For example, the word \textit{ain’t} is generally thought to be “bad grammar”, but is well attested in the history of English.

The idea that some constructions are “bad” has led to some interesting developments in the history of English. In the case of the \textit{X and me} pattern above, the idea that \textit{X and I} is to be preferred to \textit{X and me} has led to some strange reversals. For example, the first sentence type below is what is attested historically, yet the second sentence type now shows up quite often.

(2.2) Between you and me, logic is interesting.
    Between you and I, logic is interesting.

In fact, for some speakers, the avoidance of \textit{X and I} goes further, including pairs like the following.

(2.3) Logic interests you and me.
    Logic interests you and I.
Here, as with the previous pair, the first of the two sentences is what is attested historically. What’s happened is that first $X$ and $I$ extends to new environments, as in the first pair. Presumed “language authorities” decry this development and speakers “overcorrect” to avoid the supposed bad structure. This is called hypercorrection.

In all these cases, it is important to keep in mind that neither structure is intrinsically better or worse than the other. There are contextual and stylistic differences, especially in writing, but these are matters of style and custom, not a matter of goodness or badness.

### 2.1.4 Two negatives make a positive

Consider a sentence like the following:

\((2.4)\) Ernie didn’t see nothing.

Sentences of this sort are often cited as examples of the illogic of bad grammar. A sentence like this—with the interpretation that Ernie didn’t see anything—is claimed to be an instance of bad grammar. It is bad grammar because it is said to be illogical. It is taken to be illogical because there is an extra negative, didn’t and nothing, and the extra negative should make the sentence positive.

It is indeed the case that two negatives sometimes make a positive. For example. If I assert that Ernie saw no chickens when he was in the yard and you know this to be false, you might then say:

\((2.5)\) Ernie did not see no chickens when he was in the yard.

Here, the first negative not denies the second negative no. This is in contrast to the preceding example, where the first negative didn’t emphasizes the second negative nothing.

It is certainly the case that using a second negative for emphasis is stylistically marked in English. It is indicative of speech, rather than writing, and it is more casual. But is there something illogical about using a negative word for emphasis? No. Words and structures in a language can be used for many purposes and there is no particular reason why a word that has a negative meaning in one context cannot have another meaning in some other context.
In fact, this pattern is quite common in other languages. French, Spanish, and Russian all use two separate negative words to express *nothing*.

(2.6) French: Ernie n’a vu rien.
Spanish: Ernie no vio nada.
Russian: Ernie nichego ne videl.

Thus using two negative words to express a single negative idea is quite normal crosslinguistically.

It is true that the logical system we introduce in chapter 4 has the property that stacking up two negatives is equivalent to having no negatives, e.g. \((\neg\neg p \leftrightarrow p)\). This simply means that logical negation is more restricted in how it is to be interpreted than negative words in human languages.\(^1\)

### 2.1.5 I don’t have an accent

Everyone speaks their own language differently from other speakers of the same language. These differences can be in what words a person uses, how they pronounce those words, how words are combined to form sentences, etc. Some of these differences are completely idiosyncratic and reflect individual variation. Some of these differences reflect a speaker’s geographic or social origins. A person’s completely unique language is termed an *idiolect*; a set of features that reflect a particular geographic area and/or social distinction is called a *dialect*.

Let’s first look at some pronunciation differences. Some speakers in the southern US pronounce words like *pen* and *pin* the same, while most northern speakers keep these distinct. Likewise, some northern speakers make a distinction between the initial sounds of words like *witch* and *which*, while other speakers do not.

There are lots of regional differences in terms of word choice. For example, dialects vary dramatically in how they refer to a “soft drink”: *soda*, *pop*, or even, used generically, *coke*. A very interesting example is *bubbler* which means a drinking fountain. This term is only used in western Wisconsin and New England. The term apparently was the brand name for a particular drinking fountain that was sold in just those areas of the country.

\(^1\)As you might predict, our logical system has no mechanism for expressing the emphasis that the double negative sometimes expresses in English.
There are also differences in terms of word order or grammar. For example, there are dialects of English in which the following are acceptable and mean different things:

\[(2.7)\] Ernie eat it.
Ernie be eating it.

The first simply means that Ernie is eating something. The second means that Ernie habitually eats something. The latter structure is quite interesting because the distinction is not one that most dialects of English make and because the construction is stigmatized.

Another construction that shows up in some dialects is the *double modal* construction, e.g. *I might could do that*. Finally, there are a number of dialects that distinguish direct address to one person *you* vs. direct address to more than one person: *you all, y’all, you guys, yous*, etc.

The point here is that everyone’s language reflects their geographic and social origins. Different dialects exhibit different degrees of social acceptability, but there is no intrinsic qualitative difference between dialects, nor does anyone not have a dialect.

### 2.1.6 Some languages are logical

It is common to ascribe different qualities to languages. For example: German is ‘logical’, French is ‘beautiful’, or Russian is ‘harsh’. These terms have nothing to do with the language in question and typically have much more to do with our own prejudices about the people and culture.

### 2.1.7 Some languages are primitive

Various languages and dialects are often decried as ‘primitive’. Usually, this is an instance of the type of misconception just above. For example, Native American languages like Navajo or Tohono O’odham (Papago) are described as primitive languages, not because there is anything especially primitive about them, but because of unfortunate ambient attitudes that Native American cultures are primitive in some way.

Sometimes the argument is more sophisticated. For example, the dialect of English cited above where a sentence like *Ernie eat it* is acceptable is cited as primitive because it is “missing” the \(-s\) suffix that occurs in more accepted
Such arguments are specious and opportunist. Thus, in the case at hand, the dialect is cited as more primitive than other dialects because it is missing the \(-s\) suffix, yet it could, by parity of argument, be cited as less primitive than other dialects as it makes an aspectual distinction with \(be\) that other dialects do not make.

Sometimes the argument is even put in the opposite direction! Eskimo is cited as more primitive because it very quaintly has many words for snow, while English presumably has fewer. In fact, it’s been shown that Eskimo does not have any more words for snow than other languages (Pullum, 1991). Moreover, people like skiers and snowplow drivers, whose jobs or recreation depend on snow, have many more words for snow. Is their English somehow more primitive?

Occasionally, the argument is made in terms of communicative limits. For example, one might argue that French is a highly evolved language since it has words for concepts like \(\text{détente}\) or \(\text{ennui}\). This is specious too, however. French does indeed have the words cited, but the concepts are not difficult to express in words in English: “a release from tension” or “weariness and dissatisfaction”. Moreover, there are equivalent concepts that appear to be hard to express in French, but are easy in English: \(weekly, parking (lot)\), etc.

Every language will express some concepts as individual words and others as combinations of words. It’s not at all apparent that there is any rational basis to which is which in any particular language.

Finally, the character or existence of a writing system is sometimes cited as evidence of a language’s primitive nature. Thus, Navajo might be cited as “primitive” because it did not have a writing system until fairly recently, while English has had one for hundreds of years.

Writing systems are certainly a valuable piece of cultural technology, but that is probably the best way to describe them: as technology. Thus a language with a writing system has a tool at its disposal that is quite useful. What’s important to keep in mind, however, is that the presence of a writing system does not appear to correlate with any aspect of language structure.

### 2.2 Key Notions

In the previous section, we spent a fair amount of time showing what language is not. In this section, we proceed to define language in a positive fashion.
2.2.1 What is Language?

What is language? Most people would define it as some sort of communication system. This is certainly true, but it is general enough to include other communication systems as well. For example, is Morse Code an instance of language? Are traffic lights—red, yellow, and green—language?

To distinguish what we think of as language from these other systems, we need to take account of the fact that language is a more complete system. In fact, language is arguably capable of expressing virtually any idea that a human being is capable of thinking of. We can term this power *expressive completeness* and the notion will rear its head again in chapters 4 and 5.

In fact, it has occasionally been argued that language determines what we can think about. That is, it is impossible to think of things that we cannot put into words. This latter position is somewhat different from expressive completeness and highly controversial. It is referred to as the *Sapir–Whorf Hypothesis*.

Notice that expressive completeness also rules out inadvertent communication systems. For example, we might conclude that Ernie is sick from him coughing or sneezing, but we would certainly not want to characterize those as instances of language.

Expressive completeness also rules out facial gestures as an instance of language, whether those gestures be unintentional, like a smile, or deliberate, like a wink.

Thus we characterize language as an *expressively complete conventionalized communication system*. The requirement of expressive completeness rules out miniature or “toy” systems. Conventionalization rules out unintentional communication systems. Language, on this definition, is a subcase of a more general notion of communication.

Notice that this definition does not entail that a language must include a written form. In fact, the definition is neutral with respect to modality, allowing for signed languages like *American Sign Language* (ASL). The language also allows for other modalities as well, e.g. an exclusively written language.\(^2\)

Given this definition of language, we can ask whether animals have language. The question is actually a rather odd one and rests on what we mean by expressive completeness. If we mean that a language is complete with respect to any message a speaker of it might want to purvey, then surely some

---

\(^2\)This departs from the usual linguistic notion of language.
animals have language in this sense. If, on the other hand, we mean that a
language must be complete with respect to any message that we might want
to convey, then probably not, assuming that there are no animals capable
of communicating about the full range of topics that humans communicate
about.

2.2.2 Creativity

In the previous section, we showed that language learning cannot be reduced
to memorizing words and phrases. Rather, learning a language involves learn-
ing patterns and then exploiting those patterns in potentially novel ways.
Thus, for example, our knowledge about what constitutes a well-formed sen-
tence of English does not evaporate when we are confronted with a new name.
A child confronted with a new individual with a name she hasn’t heard before
is perfectly capable of uttering novel sentences describing the new individual,
e.g. Ernie is in the kitchen.

Thus knowledge of a language is knowledge of patterns.

One might believe that these patterns are simply the sensible ones, but
this would be incorrect. One piece of evidence against this idea is that the
patterns that work in a language are different from the patterns that work
in other languages. For example, we saw that some languages use double
negatives, and others do not.

Another argument against this idea comes from the fact that we have
contrasts like the following:

\[(2.8)\quad \text{Colorless green ideas sleep furiously.}
\]
\[\text{Furiously sleep ideas green colorless.}\]

Neither of these sentences makes sense in any obvious way, yet the first is
vastly more acceptable than the second. Our knowledge of English separates
the status of sentences of these types.

Both arguments support the idea that the patterns that are acceptable
in our language are not governed by what makes sense.

These patterns are quite complex. For example, most speakers of English
will prefer one of the following sentences to the other.

\[(2.9)\quad \text{Who did Ernie believe Minnie claimed that Bob saw?}
\]
\[\text{Who did Ernie believe Minnie’s claim that Bob saw?}\]
Both sentences, though long and a little complicated, make perfect sense, yet one is much better than the other. The point is that our knowledge of our language depends on knowing some rather subtle generalizations about what makes a sentence acceptable. These generalizations have to do with the structures in a sentence and not with whether the sentence makes sense in some intuitive way.

### 2.2.3 Prescriptive vs. Descriptive Grammar

Another key notion in understanding language is understanding the difference between what a speaker knows about their own language and what so-called experts tell us about language. For example, as a speaker of English, I know that I can say *You and me are studying logic*. However, as an educated English speaker, I know that we are not supposed to write such things down and instead are supposed to write: *You and I are studying logic*.

When we study what speakers actually do, we are interested in descriptive grammar. If, on the other hand, we are interested in the rules that are imposed on speakers, then we are interested in prescriptive grammar. Both are quite reasonable areas of study. The first is more about individual psychology, what it is a person actually knows about their language, albeit unconsciously. The second is more about social systems, what aspects of language are valued or not in the society at large.

### 2.2.4 Competence and Performance

An extremely important but controversial distinction is that between knowledge of a language, or competence, and the use of that language, or performance.

We can make sense of this distinction by imagining what it would be like to study language if we did not make this difference. We would study language by observing what people said. There are two problems with this.

First, people occasionally make mistakes when they speak, or occasionally change their minds in the middle of a sentence. For example, in conversation, one frequently hears sentences like these:

---

3Most prefer the first to the second.
(2.10) I think... uh, what did you say?  
Did you read the... oh yeah, now I remember.  
Ernie likes... oh, hey, Hortence!  
I bed that rook, I mean, I read that book.

Any speaker of English would recognize these as not acceptable instances of English word order, yet we utter these sorts of things all the time. If our theory of language was based purely on what we observed in the real world, we would have to account for these.

Another problem is that there are sentences that we find acceptable, yet do not utter. We’ve already treated some of these. There are acceptable, yet nonsensical sentences like the following.

(2.11) Colorless green ideas sleep furiously.

There are also sentences that refer to possible situations that we simply haven’t been confronted with:

(2.12) Hortence loves Ernie.

In this latter case, we may simply not know individuals with those names, or not have contemplated their emotional attachment.

In fact, one can argue that there are an infinite number of these sentences. Consider the following series of sentences:

(2.13) Sophie likes logic.  
Ernie knows Sophie likes logic.  
Sophie knows Ernie knows Sophie likes logic.  
Ernie knows Sophie knows Ernie knows Sophie likes logic.  
...  

Or this one:

(2.14) What is one and one?  
What is one and one and one?  
What is one and one and one and one?  
...

In each case, the series begins with a completely acceptable sentence. We can add words in a simple way up to infinity. Of course, eventually, these
become too long to understand or too long to utter before falling asleep, but there is no principled upper bound on acceptability. If we were to force ourselves to restrict our language data to only the set of observed utterances, we would completely miss the fact that the set of possible utterances is infinite.

These problems are what leads to the distinction between language competence and language performance. A science of language can then be based on either sort of data. We might choose to investigate language competence by looking at the judgments of well-formedness that a speaker is capable of. We might instead investigate language performance by looking at what type of utterances speakers actually produce and comprehend.

The distinction between competence and performance has held sway in linguistics for almost fifty years and is still generally thought to be useful. It is, however, becoming more and more controversial. An intuitive concern is that it seems suspicious to some that a science of language should be based on intuitions, rather than more “direct” data. A more substantive problem is that closer investigation of judgment data shows that those data exhibit quite a bit of variability.

2.3 General Areas

Let’s now consider the basic areas of language study. We can divide these into two broad categories. First, there are the areas that concern the actual structure of a language. Then there are the areas that concern how those structures play out in various ways.

2.3.1 Structures of Language

The areas of language study that focus on the structures of language include: phonetics, phonology, morphology, syntax, and semantics.

Phonetics

Phonetics is concerned with the sounds of language. The empirical goal is to discover what sounds are possible in language and to try to explain why only certain speech sounds occur and not others.

The method of explanation is the physical setting of speech. Patterns of speech sounds are explained in terms of how the articulatory system or the auditory system works.
Let’s consider a couple of examples of phonetic facts and phonetic explanations.

One phonetic fact that we have already mentioned is that all languages appear to include the vowel sound [a], as in the first syllable of English *father*. The usual explanation for this is that this is an extremely simple sound to produce. Oversimplifying somewhat, it is produced by just opening the mouth fully and voicing. If we assume that languages make use of sounds that are easy to produce before they make use of sounds that are hard to produce, it follows that many, if not all, languages will have [a].

Another kind of phonetic fact concerns [t]-like sounds. In English, the sound [t], as in *toe*, is produced by putting the tip of the tongue against the alveolar ridge, the bony ridge behind the upper teeth. In Spanish and Russian, on the other hand, [t] is produced by putting the tip of the tongue slightly forward, against the back of the upper front teeth.

Other languages include other [t]-like sounds where the tip of the tongue is placed further back on the roof of the mouth against the hard palate. There are no languages, however, where [t]-like sounds are produced by placing the tip of the tongue against the soft palate.

This should not be too surprising and follows directly from the physiology of articulation. The connections between the tongue and the floor of the mouth...
prevent the tip from reaching that far back (unless someone is unusually
gifted).

Let’s consider one more example. English includes the sound [v], for
example, in the word *van*. This sound is produced by bringing the upper
teeth in close proximity to the lower lip and voicing. Spanish, on the other
hand includes the sound [β], as in the word *cabo* [kaβo] ‘end’. This sound
is very similar to English [v] except that one brings the two lips in close
proximity, rather than lip and teeth.

What’s striking is that there is only one language in the world that ap-
pears to have both sounds: Ewe, spoken in Ghana and Togo. Why is this
combination so rare? The explanation is that the sounds are so very similar
that it is too hard to distinguish them. The sounds of a language tend to be
distributed so that they are maximally distinct acoustically.

Phonology

Phonology is similar to phonetics, except the focus is on the distribution
of sounds and sound patterns as instances of cognitive organization, rather
than physiology. There is therefore a natural tension between phonetics and
phonology in terms of explanation. Both disciplines deal with sounds and
generalizations about sounds, but differ in their modes of explanation.

The two disciplines differ in their methodologies as well. Phonetics is
an experimental field and makes liberal use of technology to understand the
details of articulation, audition, and acoustics. Phonetics largely focuses
on the performance of sound systems. Phonology, on the other hand, makes
much less use of technology and largely focuses on the competence underlying
sound systems.

We’ve already discussed a range of sound system facts that seem quite
amenable to phonetic explanation. What kinds of facts require a phonological
story?

One very good candidate for phonology is syllable structure. All lan-
guages parse words into syllables. In English, for example, *hat* has one
syllable, *table* two syllables, *banana* three, etc. It’s not at all clear how to
define a syllable phonetically—some phoneticians even deny that there are
syllables—so the syllable seems a reasonable candidate for a cognitive unit.
The idea is that our psychology requires that we break words up into these
units.

There are some interesting typological generalizations about how sylla-
bles work. For example, while all languages have syllables that begin with consonants, e.g. in both syllables of English happy, not all languages have syllables that begin with vowels, e.g. in both syllables of eon. So there are two broad categories of language along this dimension. First, there are languages where all syllables must begin with a consonant, e.g. Hawaiian. There are also languages where syllables can begin with either a consonant or a vowel, e.g. English. There are, however, no languages where all syllables must begin with vowels. This generalization too is thought to be a fact about our cognitive organization.

Phonology is also concerned with relations between sounds in utterances. For example, there is a process in many dialects of (American) English whereby a [t] or [d] sound is pronounced as a flap when it occurs between appropriate vowels. A flap is produced by passing the tip of the tongue quickly past the alveolar ridge and sounds much like the [r]-sound of Spanish or Russian. We will transcribe a flap like this: [r]. This process causes items to be pronounced differently in different contexts. For example, write and ride in isolation are pronounced with [t] and [d] respectively, but when they occur before a vowel, these sounds are pronounced as flaps.

\[
\text{write} \quad \text{ride}
\]

\[
\text{in isolation} \quad [\text{rayt}] \quad [\text{rayd}]
\]

\[
\text{before a vowel-initial word} \quad [\text{rayr}] \text{ a letter} \quad [\text{rayr}] \text{ a horse}
\]

\[
\text{before a vowel-initial suffix} \quad [\text{rayr}]\text{er} \quad [\text{rayr}]\text{er}
\]

Not all languages do this and so this cannot simply be a function of the physiology. Characterizing these sorts of generalizations and the search for explanation in the domain of cognition are part of phonology.

**Morphology**

Morphology is concerned with the combination of meaningful elements to make words. It can be opposed to phonology, which we can characterize as the combination of meaningless elements to make words. For example, a word like hat is composed of three sounds, three meaningless elements: [h], [æ], and [t]. There is only one meaningful element: the word itself. A word like unhappy has a complex phonology—it has six sounds: [ʌ], [n], [h], [æ], [p], [i]—and a complex morphology: it is composed of two meaningful elements: un- and happy. The element un- is a prefix, an element that
cannot occur alone, but can be attached to the left of another element. It expresses negation. The element *happy* is a stem and can occur alone.

There are also suffixes. For example, the word *books* is composed of a stem *book* and a suffix *–s*, which expresses plurality. Prefixes, suffixes, stems, etc. are called *morphemes*.

The most important thing to keep in mind about morphology is that it can, in some cases, be boundless. Hence the set of possible words in a language with boundless morphology is infinite. Here is an example from English where a set of suffixes can be attached without bound.

\[(2.17)\]
\[
\begin{array}{c}
nation \\
national \\
nationalize \\
nationalization \\
nationalizational \\
nationalizationalize \\
\ldots
\end{array}
\]

Another extremely important point is that elements—*morphemes*—are not combined in a linear fashion, but are nested. For example, a word like *nationalize* has three morphemes that are grouped together as represented in the tree below.

\[(2.18)\]
\[
\begin{array}{c}
nation \\
nationalize
\end{array}
\]

This structure can be important to the meaning of a word. Consider a compound word like *budget bottle brush*. This has two different meanings associated with two different structures.

\[(2.19)\]
\[
\begin{array}{c}
budget \\
bottle \\
brush \\
budget \\
bottle \\
brush
\end{array}
\]

The structure on the left is associated with the meaning where the combination refers to an inexpensive brush for bottles; the structure on the right is associated with the meaning where it refers to a brush for inexpensive bottles.
The two ideas come together in the following examples. A word like *unforgivable* has a single meaning: not able to be forgiven. A word like *unlockable* actually has two meanings: not able to be locked and able to be unlocked. The ambiguity of the second word correlates with the two different possible structures for it.

(2.20)\[
\text{un} \quad \text{lock} \quad \text{able} \quad \text{un} \quad \text{lock} \quad \text{able}
\]

The word *unforgivable* only has one meaning because only one structure is possible:

(2.21)\[
\text{un} \quad \text{forgiv} \quad \text{able}
\]

No other meaning is possible because the other structure below is not possible.

(2.22)\[
\text{un} \quad \text{forgiv} \quad \text{able}
\]

The reason this latter tree is not possible is that to produce it, we would have to first combine *un-* and *forgive* into *unforgive*, and that is not a word of English.

**Syntax**

Syntax is concerned with how words are combined to make sentences. We’ve already cited examples in sections 2.1 and 2.2 above that establish key properties we’ve ascribed to language. First, syntax showed us that language learning is more than memorization. Second, it established that knowledge of language is knowledge of the generalizations that underlie what is well-formed in the language. Third, it established that what is well-formed in a language is not determined by what “makes sense”.

A remaining essential point is that words are not combined in a linear fashion. Rather, as with morphology, words are combined in a nested fashion. Consider, for example, a sentence like the following:

(2.23) Ernie saw the man with the binoculars.

There are two possible meanings for this sentence. First, it could be that Ernie used binoculars to see the man. Alternatively, the man has binoculars and Ernie saw him. These two meanings are based on two different structures. The first has the man and with the binoculars as sisters within a larger grouping, labeled VP here for Verb Phrase.

(2.24)

Ernie  VP
  /\    /\ 
 /saw/ /NP/ 
 |  |  |
the man with the binoculars

The other structure groups with the binoculars directly within the same phrase as the man, labeled NP here for Noun Phrase.

(2.25)

Ernie  VP
  /\    /\ 
 /saw/ /NP/ 
 |  |  |
the man with the binoculars
In the first structure, *with the binoculars* modifies the verb; in the second structure, it modifies the noun.

The point is that words are grouped together into structures and those structures contribute to the meaning of sentences. Syntactic competence includes knowledge of what structures are possible in the formation of sentences.

**Semantics**

Semantics is concerned with the meanings of words and sentences. One kind of semantics deals with how the meaning of a sentence is computed from its words and the way those words are grouped together. As we have already shown, the groupings can make dramatic contributions to the meaning of a sentence or of a word, as in the *binocular* example above. We also considered examples where there are multiple meaning differences that are quite subtle, but are not superficially associated with structural differences: the example treated in chapter 1 of *Two women saw two men*.

We will see that logic, treated in chapters 4 and 5, is quite useful in semantics.

### 2.3.2 Other Areas

There are a number of other really interesting areas of language study and we list just a few of them here (in no particular order).

**Psycholinguistics** is the study of performance, how language is actually used. The methodology is typically experimental. Psycholinguists study language production, language comprehension, speech perception, lexical access, etc.

**Neurolinguistics** is concerned with language in the brain. At the theoretical level, it focuses on how language is processed in actual brain structures. At a more applied level, it deals with various sorts of cognitive disorders involving language.\(^4\)

---

4. There is something called *Neurolinguistic Programming*, but this is a misnomer. This is not a field of language study. Occasionally, one sees the abbreviation NLP for this field. NLP is also used for *Natural Language Processing*, which is a field of language study.
Sociolinguistics is concerned with the relationship between language and society. It is concerned with how social forces are reflected in language and with how language affects social variables.

Language Acquisition One can study how language is acquired. This is done in several ways: following the development of individual children or running experiments on sets of children.

Writing Systems can be studied for their own intrinsic interest, but also as a window into the structure of a language or as evidence for how language changes over time.

Literature also provides an interesting vantage point on language. Again, it can be studied for its own intrinsic interest, but also for the information it provides about the structure of language.

Applied Linguistics refers to several different fields, all of which use the study of language in some concrete application. These include, for example, language teaching and forensic linguistics.

Computational Linguistics includes two broad domains. One is the use of language in computational tasks, e.g. machine translation, text mining, speech understanding, speech synthesis, etc. The other area is the mathematical modeling of language. Many of the foundational areas we treat in this book are part of this latter area.

Discourse Analysis is the investigation of how units larger than a sentence are constructed. This can include conversation or texts. The latter can bring this area close to the study of literature.

Historical Linguistics is the investigation of language change. This can be done by comparing modern languages, looking at historical records, or looking at the language-internal residue of historical changes.

2.4 Summary

This chapter has introduced the study of language generally. We began with a refutation of some of the more egregious misconceptions about language.

We then established a number of key properties of language. These include a definition of language as an expressively complete conventionalized
system of communication. We also argued at several points that the set of words and sentences is infinite and that knowledge of language is more than just knowing what those words and sentences are.

Finally, we reviewed the main areas of language structure and many of the areas of language study.

2.5 Exercises

1. We’ve seen that languages are sometimes characterized in almost anthropomorphic terms, e.g. “harsh”, “beautiful”, “logical”, etc. Explain why.

2. We’ve seen that sometimes a concept that is encoded with a word in one language is expressed with a phrase in another language. Why do you think this happens?

3. Does the difference between competence and performance apply to written language? Explain why or why not.

4. We’ve cited several structures in English that show that the set of words and the set of sentences are infinite. Find another that’s different from the ones cited in the text.

5. Find a description of how the “nothing” construction works in some language we have not discussed. Describe the pattern and provide some examples. Does the language have double negation?
Chapter 3

Set Theory

What is Set Theory and why do we care? Set Theory is—as we would expect—the theory of sets. It’s an explicit way of talking about elements, their membership in groups, and the operations and relationships that can apply between elements and groups.

Set Theory is important to language study for several reasons. First, it is even more foundational than all the other topics we cover subsequently. That is, many of the other topics we will treat are grounded in set-theoretic terms.

A second reason set theory is important to know about is that there are language issues that can be treated directly in terms of set theory, e.g. features, issues of semantic entailment, and constraint logic.

3.1 Sets

A set is an abstract collection or grouping of elements. Those elements can be anything, e.g. words, sounds, sentences, affixes, etc. In the following, we will represent the names of sets in all capital letters: $A$, $B$, $C$, etc.

Sets can be defined in several different ways. The simplest is to simply list the members of the set. For example, we might define the set $A$ as being composed of the elements $x$, $y$, and $z$. This can be expressed as:

\[(3.1) \ A = \{x, y, z\}\]

The ordering of elements in the curly braces is irrelevant; a set is defined
by what elements it contains, not by any ordering or priority among those elements. Thus the following are equivalent to the preceding.

\[(3.2) \quad A = \{x, z, y\}\]
\[A = \{y, x, z\}\]
\[A = \{y, z, x\}\]
\[A = \{z, x, y\}\]
\[A = \{z, y, x\}\]

Notice too that it makes no sense to repeat an element. Thus the set \(A = \{a, b, c\}\) is the same as \(A = \{a, b, c, b\}\).

As above, the elements of a set can be anything. For example:

\[(3.3) \quad A = \{æ, ñ, n\}\]
\[B = \{\text{French, O’odham, Abkhaz, Spanish}\}\]
\[C = \{ñ, 78, \text{The Amazon}\}\]

Sets can also be defined by the properties their elements bear. For example, the set of nasal consonants in English is defined as the set of consonants in which air flows out the nose: \([m]\), \([n]\), and \([ñ]\) in \(\text{tam} \ [\text{tam}]\), \(\text{tan} \ [\text{tan}]\), and \(\text{tang} \ [\text{tæn}]\), respectively. The set can be defined by listing the elements:

\[(3.4) \quad A = \{m, n, ñ\}\]

or by specifying that the elements of the set are those elements that bear the relevant properties:

\[(3.5) \quad A = \{x \mid x \text{ is a nasal consonant of English}\}\]

This expression is read as: \(A\) is composed of any \(x\), where \(x\) is a nasal consonant of English.

Similar sets can be defined in morphology. We can define the set composed of the singular members of the present tense indicative mood of the Spanish verb \(\text{cantar}\) to ‘sing’ in the same two ways:
(3.6) \( A = \{canto, cantas, canta\} \)

(3.7) \( A = \{x \mid x \text{ is a singular form of the present tense of } cantar.\} \)

Be careful though. Defining a set in terms of the properties of its elements takes us onto dangerous ground. We want to be as explicit as possible in specifying what can qualify as a property that can be used to pick out the elements of a set.

The third way to define sets is by recursive rules. For example, the set of positive integers \((1, 2, 3, \ldots)\) can be defined as follows:

1. \(1\) is a positive integer.
2. If \(n\) is a positive integer, then so is \(n + 1\).
3. Nothing else is a positive integer.

This method of defining a set has several interesting properties. First, notice that the set thus defined is infinite in size. It has an unbounded number of elements. Second, a recursive definition typically has three parts, just as this one does. First, there is a base case, establishing directly the membership of at least one element. Second, there is the recursive case, establishing that additional members of the set are defined in terms of elements we already know are members. Finally, there is a bounding clause, limiting membership to the elements accommodated by the other two clauses.

This may seem excessively “mathematical”, but the same method must be used to define “possible word” and—in a more sophisticated form—must be used to define “possible sentence”. For example, to get a sense of this, how would you define the set of all possible strings using the letters/sounds \([a]\) and \([b]?)^1\ (Assume that any sequence of these is legal.)

Finally, we define the size of some set \(A\) as \(|A|\). If, for example, \(A = \{x, y, z\}\), then \(|A| = 3\).

\(^1\)This is given as an exercise at the end of the chapter.
CHAPTER 3. SET THEORY

3.2 Membership

If an element $x$ is a member of the set $A$, we write $x \in A$; if it is not, we write $x \notin A$. The membership relation is one between a potential element of a set and some set.

We have a similar relation between sets: subset.

**Definition 1 (Subset)** Set $A$ is a subset of set $B$ if every member of $A$ is also a member of $B$.

Notice that subset is defined in terms of membership. For example, given the set $A = \{y, x\}$ and the set $B = \{x, y, z\}$, the former is a subset of the latter—$A \subseteq B$—because all members of the former—$x$ and $y$—are members of the latter. If set $A$ is not a subset of set $B$, we write $A \nsubseteq B$. Notice that it follows that any set $A$ is a subset of itself, i.e. $A \subseteq A$.

There is also a notion of **proper subset**.

**Definition 2 (Proper Subset)** Some set $A$ is a proper subset of some set $B$ if all elements of $A$ are elements of $B$, but not all elements of $B$ are elements of $A$.

In this case, we write $A \subset B$. If $A$ is not a proper subset of $B$, we write $A \nsubset B$.

The difference between the notions of subset and proper subset is the possibility of **identity**. Sets are identical when they have the same members.

**Definition 3 (Set extensionality)** Two sets are identical if and only if they have the same members.

To say that some set $A$ is a subset of some set $B$ says that either $A$ is a proper subset of $B$ or $A$ is identical to $B$. When two sets are identical, we write $A = B$. It then follows that if $A \subseteq B$ and $B \subseteq A$, that $A = B$.

Note that $\subset$ and $\in$ are different. For example, $\{a\}$ is a subset of $\{a, b\}$, but not a member of it, e.g. $\{a\} \subseteq \{a, b\}$, but $\{a\} \notin \{a, b\}$. On the other hand, $\{a\}$ is a member of $\{\{a\}, b, c\}$, but not a subset of it, e.g. $\{a\} \in \{\{a\}, b, c\}$, but $\{a\} \nsubseteq \{\{a\}, b, c\}$.

We also have the empty set $\emptyset$, the set that has no members. It follows that the empty set is a subset of all other sets. Finally, we have the power set of $A$: $2^A$ or $\wp(A)$: all possible subsets of $A$. The designation $2^A$ reflects
the fact that a set with $n$ elements has $2^n$ possible subsets. For example, the set \{x, y, z\} has $2^3 = 8$ possible subsets.\(^2\)

\[(3.8) \quad \{x, y, z\} \quad \{x, y\} \quad \{x\} \quad \emptyset \quad \{x, z\} \quad \{y\} \quad \{y, z\} \quad \{z\}\]

### 3.3 Operations

There are various operations that can be applied to sets. The union operation combines all elements of two sets. For example, the union of $A = \{x, y\}$ and $B = \{y, z\}$ is $A \cup B = \{x, y, z\}$. Notice how—as we should expect—elements that occur in both sets only occur once in the new set. One way to think of union is as disjunction. If $A \cup B = C$, then an element that occurs in either $A$ or $B$ (or both) will occur in $C$.

The other principal operation over sets is intersection, which can be thought of as conjunction. For example, the intersection of $A = \{x, y\}$ and $B = \{y, z\}$ is $A \cap B = \{y\}$. That is, if an element is a member of both $A$ and $B$, then it is a member of their intersection.

We have already defined the empty set $\emptyset$: the set that contains no elements. We can also define the universal set $U$: the set that contains all elements of the universe. Similarly, we can define difference and complement. Thus $A'$, the complement of $A$, is defined as all elements that are not in $A$. That is, all elements that are in $U$, but not in $A$. For example, if $A = \{x, y\}$ and $U = \{x, y, z\}$, then $A' = \{z\}$.

Set complement and intersection can be used to define difference. The difference of two sets $A$ and $B$ is defined as all elements of $A$ minus any elements of $B$, i.e. $A - B$. This can also be expressed as the intersection of $A$ with the complement of $B$, i.e. $A \cap B'$. Likewise, the complement of $A$, $A'$, can be defined as $U - A$. Notice that not all elements of $B$ need to be in $A$ for us to calculate $A - B$. For example, if $A = \{x, y\}$ and $B = \{y, z\}$, then $A - B = \{x\}$.

\(^2\)Recall that $2^3 = 2 \times 2 \times 2 = 8$. 
3.4 Fundamental Set-theoretic Equalities

There are a number of beautiful and elegant properties that hold of sets, given the operations that we’ve discussed. We go through a few of these in this section.

*Idempotency* has it that when a set is unioned or intersected with itself, nothing happens.

\[(3.9) \text{ Idempotency} \]
\[X \cup X = X \]
\[X \cap X = X \]

*Commutativity* expresses that the order of arguments is irrelevant for union and intersection.

\[(3.10) \text{ Commutativity} \]
\[X \cup Y = Y \cup X \]
\[X \cap Y = Y \cap X \]

*Associativity* holds that the order with which sets are successively unioned or successively intersected is irrelevant.

\[(3.11) \text{ Associativity} \]
\[(X \cup Y) \cup Z = X \cup (Y \cup Z) \]
\[(X \cap Y) \cap Z = X \cap (Y \cap Z) \]

*Distributivity* governs the interaction between union and intersection.

\[(3.12) \text{ Distributivity} \]
\[X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z) \]
\[X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z) \]

*Identity* governs how the universal set $U$ and the empty set $\emptyset$ can be intersected or unioned with other sets.
CHAPTER 3. SET THEORY

(3.13) Identity
\[ X \cap U = X \]
\[ X \cup \emptyset = X \]

*Domination* includes several relationships rather similar to *Identity*.

(3.14) Domination
\[ X \cap \emptyset = \emptyset \]
\[ X \cup U = U \]

The *Complement Laws* govern complements and differences.

(3.15) Complement Laws
\[ X \cup X' = U \]
\[ X \cap X' = \emptyset \]

The *Double Complement Law* says that the complement of the complement of a set is the same as the original set.

(3.16) Double Complement Law
\[ (X')' = X \]

We also have the *Relative Complement Law*, which is basically a definition of set difference.

(3.17) Relative Complement Law
\[ X - Y = X \cap Y' \]

Finally, *DeMorgan’s Law* shows how complement allows us to define intersection and union in terms of each other.

(3.18) DeMorgan’s Law
\[ (X \cup Y)' = X' \cap Y' \]
\[ (X \cap Y)' = X' \cup Y' \]
CHAPTER 3. SET THEORY

When we get to the Laws of Sentential Logic in Section 4.7, we will see a lot of similarity with these.

3.5 Theorems

Let's try to understand and prove some things in set theory. Here are a few theorems of set theory.

1. If $A \subseteq B$ then $A \cup C \subseteq B \cup C$
2. If $A \subseteq B$ then $A \cap C \subseteq B \cap C$
3. If $A \subseteq B$ then $C - B \subseteq C - A$
4. $A \cap (B - A) = \emptyset$

Let's consider how we might prove the first one. We can do this informally at this stage using what we will call Indirect Proof in section 4.10.

First, let's make sure we understand the theorem by stating it in words. If the set $A$ is a subset of the set $B$, then the union of $A$ with any set $C$ is a subset of the union of $B$ with the same set $C$. A more informal rewording might make this more intuitive. If $A$ is a subset of $B$, then if you add some elements to $A$ and add the same elements to $B$, then $A$ is still a subset of $B$.

Let's look at an example. Let's define $A = \{a, b\}$, and $B = \{a, b, c\}$. The first set is certainly a subset of the second, $A \subseteq B$, as every member of $A$ is also a member of $B$.

Let's now union $A$ and $B$ with some other set $C = \{g, h\}$. We now have that $A \cup C = \{a, b, g, h\}$ and $B \cup C = \{a, b, c, g, h\}$. The union operation has added the same elements to both sets, so the first unioned set is still a subset of the second unioned set: $A \cup C \subseteq B \cup C$.

We will make use of a proof technique called Indirect Proof, or Proof by Contradiction. The basic idea is that if you want to prove some statement $S$, you show instead that if $S$ were not true, a contradiction would result. This sounds quite formal, but we use this technique all the time. Imagine that $S$ is some political candidate you are committed to and you are arguing to a friend that they should vote for $S$ too. You might very well try to make the case by making dire predictions about what would happen if $S$ is

---

3 We will consider proof and proof strategies in more detail in chapters 4 and 5.
not elected, or by enumerating the shortcomings of the candidates that $S$ is running against. These are both instances of essentially the same technique.

In the case at hand, we will attempt to prove a contradiction from these two statements: $A \subseteq B$ and $A \cup C \not\subseteq B \cup C$. If the latter is true, then there must be at least one element, call it $x$, that is a member of $A \cup C$, but is not a member of $B \cup C$. If it is a member of $A \cup C$, then by the definition of union, it must be a member of $A$ or $C$ or both. If it is a member of $A$, then it must be a member of $B \cup C$ because all members of $A$ are members of $B$ and all members of $B$ are, by the definition of union, members of $B \cup C$. Thus $x$ cannot be a member of $A$. It must then be a member of $C$. However, if it is a member of $C$, then it must be a member of $B \cup C$, because, by the definition of union, all members of $C$ are members of $B \cup C$. Hence $x$ cannot be a member of $C$.

It now follows that there can be no element $x$. This must be the case because $x$ would have to be a member of $A$ or $C$ and neither is possible. Hence, we have a contradiction and it cannot be the case that $A \subseteq B$ and $A \cup C \not\subseteq B \cup C$.

4 We leave the remaining theorems as exercises.

3.6 Ordered Pairs, Relations, and Functions

Sets have no order, but we can define groups with an ordering, e.g. an ordered pair. For example, if we want to say that $x$ and $y$ are in an ordered pair, we write $\langle x, y \rangle$. We’ve already seen that, for example, $\{x, y\} = \{y, x\}$, but $\langle x, y \rangle \neq \langle y, x \rangle$. Ordered groups with more elements are defined in the obvious way, e.g. $\langle x, y, z, w \rangle$, etc. Notice that, while $\{x, x\}$ makes no sense, $\langle x, x \rangle$ does.

Ordered pairs may seem like something that should be “outside” set theory, but they can be defined in set-theoretic terms as follows. An ordered pair $\langle x, y \rangle$ is defined as the set composed of the set of its first member plus the set composed of both members: $\{\{x\}, \{x, y\}\}$. Larger ordered groups can be defined recursively. Thus $\langle x, y, z \rangle$ is defined as $\langle \langle x, y \rangle, z \rangle$.

A relation $R$ pairs elements of one set $A$ with elements of the same set $A$ or a different set $B$. If the relation $R$ pairs element $a$ to element $b$, then we write $aRb$ or, for legibility, $R(a, b)$. For example, we might define a relation

\footnote{All proofs in the text will be marked with a subsequent box.}
between active sentences and passive sentences, or one that relates oral vowels to nasal ones, e.g. $aR\tilde{a}$.

We can define a relation as a set of ordered pairs. If, for example, $aRb$ and $cRd$, then we can represent these as $\langle a, b \rangle \in R$ and $\langle c, d \rangle \in R$. If these are all the pairs that $R$ gives us, then we can represent $R$ as $R = \{\langle a, b \rangle, \langle c, d \rangle\}$.

We can also have $n$-place relations, where two or more elements are paired with another element. These are of course ordered triples: $\langle a, b, c \rangle$, and the whole relation is the set of these triples.

The domain of a two-place relation is defined as the first member of the ordered pair and the range is defined as the second member of the pair.

Let’s adopt some additional notation. If some relation $R$ pairs every element of some set $A$ with every element of the same set, then we write $R = A \times A$. Thus, if $R = A \times A$ and $A = \{x, y, z\}$, then $R = \{\langle x, x \rangle, \langle x, y \rangle, \langle x, z \rangle, \langle y, x \rangle, \langle y, y \rangle, \langle y, z \rangle, \langle z, x \rangle, \langle z, y \rangle, \langle z, z \rangle\}$. We can see that $|R| = |A| \times |A| = 3 \times 3 = 9$.

Since relations can pair elements of different sets, the size of the relation varies accordingly. For example, if $R = A \times B$ and $A = \{x, y, z\}$ and $B = \{a, b\}$, then $R = \{\langle x, a \rangle, \langle x, b \rangle, \langle y, a \rangle, \langle y, b \rangle, \langle z, a \rangle, \langle z, b \rangle\}$. In this case, $|R| = |A| \times |B| = 2 \times 3 = 6$.

The complement $R'$ of a relation $R$ is defined as every pairing over the sets that is not included in $R$. Thus if $R \subseteq A \times B$, then $R' =_\text{def} (A \times B) - R$. For example, if $R$ relates $A = \{x, y, z\}$ and $B = \{a, b\}$ (that is $R \subseteq A \times B$), and $R = \{\langle x, a \rangle, \langle x, b \rangle, \langle y, a \rangle, \langle y, b \rangle\}$, then $R' = \{\langle z, a \rangle, \langle z, b \rangle\}$.

The inverse $R^{-1}$ of a relation $R$ reverses the domain and range of $R$. Thus, if $R \subseteq A \times B$, then $R^{-1} \subseteq B \times A$. For example, if $R = \{\langle x, a \rangle, \langle x, b \rangle, \langle y, a \rangle, \langle y, b \rangle\}$, then $R^{-1} = \{\langle a, x \rangle, \langle b, x \rangle, \langle a, y \rangle, \langle b, y \rangle\}$.

A function is a special kind of relation where every element of the domain is paired with just one element of the range. For example, if the function $F$ pairs $a$ with $b$, we write $F(a) = b$. If the domain of $F$ is $A = \{x, y\}$ and the range is $B = \{a, b\}$, then $F \subseteq A \times B$. It follows from the definition of a function, that $F$ can only be one of the following:

\begin{align*}
3.19 & \quad a. & F &= \{\langle x, a \rangle, \langle y, a \rangle\} \\
& \quad b. & F &= \{\langle x, b \rangle, \langle y, b \rangle\} \\
& \quad c. & F &= \{\langle x, a \rangle, \langle y, b \rangle\} \\
& \quad d. & F &= \{\langle x, b \rangle, \langle y, a \rangle\}
\end{align*}
There is no other function from $A$ to $B$.

Finally, relations exhibit a variety of properties that are quite useful.

**Reflexivity** A relation $R$ is reflexive if and only if, for every element $x$ in $R$, $\langle x, x \rangle \in R$.

**Symmetry** A relation $R$ is symmetric when $\langle x, y \rangle \in R$ if and only if $\langle y, x \rangle \in R$.

**Transitivity** A relation $R$ is transitive if and only if for all pairs $\langle x, y \rangle$ and $\langle y, z \rangle$ in $R$, $\langle x, z \rangle$ is in $R$.

The terminology gets heinous when we begin to consider the way these properties might *not* hold of some relation. For example, a relation that contains no instances of $\langle x, x \rangle$ is *irreflexive*, but if it is simply missing *some* instances of $\langle x, x \rangle$, then it is *nonreflexive*.

### 3.7 Language examples

Let’s now look at some simple examples of these notions in the domain of language.

#### 3.7.1 Examples of Sets

Let’s start with the words of English. This is, of course, a set, e.g. $W = \{ \text{run, Ernie, of, ...} \}$. Any word of English will be a member of this set. For example: $\text{hat} \in W$.

The verbs of English are also a set: $V = \{ \text{run, sat, envelops, ...} \}$. As we would expect, the verbs of English are a subset of the words of English, $V \subseteq W$, and, in fact, a proper subset: $V \subset W$.

Let’s assume that nouns are also a set and a proper subset of the set of English words: $N \subset W$. The union of nouns and verbs, $N \cup V$, would then be the set of any word that is a noun or a verb. Is this true: $N \cup V \subset W$? What is the intersection of $N$ and $V$, i.e. $N \cap V$? Can you give an example? If some word is a member of $N$, does it follow that it is also a member of $N \cup V$? Of $N \cap V$?

We can also use set theory to characterize syntax or morphology. For example, a sentence like *Ernie likes logic* can be treated as an ordered triple:
⟨Ernie, likes, logic⟩. Why would it be a mistake to view sentences as normal sets?

We can go further, in fact. Recall that a speaker’s knowledge of what constitutes a well-formed sentence is best characterized in terms of hierarchical structures. Thus a sentence isn’t a flat string of words, but organized into nested phrases. We can incorporate this by viewing a sentence not as a flat ordered tuple, but as composed of nested tuples. For example, if we were to say that likes logic is a VP, we could represent Ernie likes logic as: ⟨Ernie, (likes, logic)⟩.

3.7.2 Examples of Relations and Functions

Relations are ubiquitous in language. For example, we can posit a relation \( R \) between (unmarked) verbs in the present tense:

\[
\{\text{jump}, \text{break}, \text{finish}, \text{ring}, \ldots\}
\]

and verbs marked for the past tense:

\[
\{\text{jumped}, \text{broke}, \text{finished}, \text{rang}, \ldots\}
\]

Thus:

\[
(3.22) \quad R = \left\{ \langle \text{jump}, \text{jumped} \rangle, \langle \text{break}, \text{broke} \rangle, \langle \text{finish}, \text{finished} \rangle, \langle \text{ring}, \text{rang} \rangle, \ldots \right\}
\]

The inverse of this is then:

\[
(3.23) \quad R^{-1} = \left\{ \langle \text{jumped}, \text{jump} \rangle, \langle \text{broke}, \text{break} \rangle, \langle \text{finished}, \text{finish} \rangle, \langle \text{rang}, \text{ring} \rangle, \ldots \right\}
\]
Is $R$ a function? Recall, that a relation must satisfy two properties to be a relation. First, does every verb of English have a past tense? Yes. Second, is there one and only one past tense for every verb? This is tricky.

First, there are verbs where the past tense form looks just like the present tense for, e.g. $R(\text{hit, hit})$. On the other hand, there are verbs that seem to have more than one past tense form. For example, the verb $\text{ring}$ has the past tense forms $\text{rang}$ and $\text{ringed}$. These seem to have different meanings, however.

(3.24) The bell rang.
The wall ringed the city.

To accommodate cases like this, we might want to say that there are two verbs $\text{ring}_1$ and $\text{ring}_2$, each with unique past tense forms.

Another problem, however, is that there are verbs like $\text{dive}$ where multiple past tense forms are possible: $\text{dived}$ and $\text{dove}$. Some speakers prefer one, some the other, and some vacillate. If $R(\text{dive, dove})$ and $R(\text{dive, dived})$, then $R$ is not a function.

On the other hand, the $\text{dive}$ problem does not prevent $R^{-1}$ from being a function.

Let’s consider some relations that are relevant to sound systems. For example, we might posit a relation $R$ from words to numbers that says how many letters are in a word, e.g. $R(\text{hat, 3})$, $R(\text{orange, 6})$, $R(\text{appropriate, 11})$. We could also do this with respect to the number of sounds in a word: $R([\text{hæt}], 3)$, $R([\text{ɔræŋ}], 5)$, $R([\text{ɑ̃prɔpri@t}], 9)$. This is clearly a function. Every single word has one and only one length. The inverse $R^{-1}$ is not a function as many words have the same length.

We might also posit a relation $L$ that indicates that the first element is longer than the second element, again, in either letters or sounds. Thus, if we define $L$ in terms of letters, we get instances like these: $L(\text{orange, hat})$, $L(\text{appropriate, hat})$, $L(\text{appropriate, orange})$, etc. Notice that neither this relation nor its inverse are functions.

Let’s consider now whether any of these relations exhibit the properties we discussed above. First, none of them are reflexive. Recall that for a relation to be reflexive, every element in the domain must be paired with itself. An example of a reflexive relation might be the notion of at least as long as: $L_1$. We would then have $L_1(\text{hat, hat})$, $L_1(\text{orange, orange})$, etc.

Only the last relation is transitive. For example, from the fact that the
word *orange* is longer than the word *hat*, \( L(orange, hat) \), and the fact that the word *appropriate* is longer than the word *orange*, \( L(appropriate, orange) \), it follows that the word *appropriate* is longer than the word *hat*. That is: \( L(appropriate, hat) \).

Finally, none of the relations are symmetric. Recall that for a symmetric relation, the domain and range are interchangeable. An example of a symmetric relation might be the notion of *the same length as*: \( L_2 \). Thus if the word *hat* is the same length as the word *pan*, \( L_2(hat, pan) \), then it follows that the word *pan* is the same length as the word *hat*: \( L_2(pan, hat) \).

### 3.8 Summary

This chapter introduced basic set theory. We began with a definition of what a set is and provided three mechanisms for defining a set: by enumeration, by properties, and by recursive rules. We also defined a notion of set size such that \(|A|\) represents the size of some set \( A \).

We went on to consider notions of set membership. An element \( x \) can be a member of a set \( A \), e.g. \( x \in A \). Membership allows us to define the relations subset and proper subset between sets. We also defined the very important notion of set extensionality which holds that two sets are identical if they share the same members. In general terms, this means that sets are defined solely by their members.

We defined three special sets. First, there is the empty set \( \emptyset \) which has no members. There is also the universal set \( U \), which has *all* members.\(^5\) Finally, we defined the notion of a power set of any set \( A \), \( 2^A \) or \( \wp(A) \), which is the set of all possible subsets of \( A \).

There are several different operations which can be applied to sets. Union merges two sets. Intersection finds the overlap of two sets. Complement finds everything that is *not* in some set, and difference removes the elements of one set from another.

We presented a number of general laws about sets: Idempotency, Commutativity, Associativity, Distributivity, Identity, Domination, Complement Laws, Double Complement Law, Relative Complement Law, and DeMorgan’s Law. We also showed a few theorems about set theory and showed how we might prove them using indirect proof.

\(^5\)This is actually a very tricky notion. For example, if the universal set contains all members, then does it contain itself?
Finally, we developed a notion of ordered tuple and use it to define relations and functions. We showed what inverse and complement relations are and defined reflexivity, transitivity, and symmetry with respect to relations.

### 3.9 Exercises

1. Provide a recursive definition of the set of all possible sequences of $a$ and $b$.

2. If $A = \{a, a, a, a\}$, what is $|A|$?

3. As we noted on page 34, it follows that if $A \subseteq B$ and $B \subseteq A$, that $A = B$. Explain why this is so. Prove that this is so.

4. It follows that the empty set is a subset of all other sets. Explain why.

5. The empty set is a proper subset of all sets except one. Which one and why?

6. We gave a number of theorems on page 38. Prove these.

7. Ordered groups larger than two are defined recursively in terms of pairs. Thus $\langle x, y, z \rangle$ is defined as $\langle \langle x, y \rangle, z \rangle$. How do we encode this with simple sets?

8. Use set theory to characterize the different meanings we discussed in chapter 1 for “Two women saw two men.”

9. Why would it be a mistake to formalize sentences directly in terms of sets?

10. Find a symmetric relation in the syntax of a language that is not your native language (or English), and that is different from the examples discussed in the text. Show how it works.

11. Give an example of a proper subset relation in language different from the ones discussed in the text.
Chapter 4

Sentential Logic

In this chapter, we treat sentential logic, logical systems built on sentential connectives like and, or, not, if, and if and only if. Our goals here are threefold:

1. To lay the foundation for a later treatment of full predicate logic.
2. To begin to understand how we might formalize linguistic theory in logical terms.
3. To begin to think about how logic relates to natural language syntax and semantics (if at all!).

4.1 The intuition

The basic idea is that we will define a formal language with a syntax and a semantics. That is, we have a set of rules for how statements can be constructed and then a separate set of rules for how those statements can be interpreted. We then develop—in precise terms—how we might prove various things about sets of those statements.

4.2 Basic Syntax

We start with an finite set of letters: \( \{p, q, r, s, \ldots \} \). These become the infinite set of atomic statements when we add in primes, e.g. \( p', p'', p''', \ldots \). This infinite set can be defined recursively.
Definition 4 (Atomic statements) Recursive definition:

1. Any letter \( \{p, q, r, s, \ldots \} \) is an atomic statement.

2. Any atomic statement followed by a prime, e.g. \( p', q'', \ldots \) is an atomic statement.

3. There are no other atomic statements.

The first clause provides for a finite number of atomic statements: 26. The second clause is the recursive one, allowing for an infinite number of atomic statements built on the finite set provided by the first clause. The third clause is the limiting case.

Using the sentential connectives, these can be combined into the set of well-formed formulas: WFFs. We also define WFF recursively.

Definition 5 (WFF) Recursive definition:

1. Any atomic statement is a WFF.

2. Any WFF preceded by ‘\( \neg \)’ is a WFF.

3. Any two WFFs can be made into another WFF by writing one of these symbols between them, ‘\( \land \)’, ‘\( \lor \)’, ‘\( \to \)’, or ‘\( \leftrightarrow \)’, and enclosing the result in parentheses.

4. Nothing else is a WFF.

The first clause is the base case. Notice that it already provides for an infinite number of WFFs since there are an infinite number of atomic statements. The second clause is recursive and provides for an infinite number of WFFs directly, e.g. \( \neg p, \neg \neg p, \neg \neg \neg p \), etc. The third clause is also recursive and thus also provides for an infinite number of WFFs. Given two WFFs \( p \) and \( q \), it provides for \( (p \land q) \), \( (p \lor q) \), \( (p \to q) \), and \( (p \leftrightarrow q) \). The fourth clause is the limiting case. The second and third clauses can be combined with each other to produce infinitely large expressions. For example:

\[ (4.1) \quad (p \to \neg \neg (\neg p \lor (p \lor p))) \]
\[ \quad ((p \to p) \leftrightarrow (p \land q)) \]
\[ \quad (p \land (q \lor (r \land (s \lor t)))) \]
\[ \ldots \]
Be careful with this notation. The point of it is precision, so we must be precise in how it is used. For example, parentheses are required for the elements of the third clause and disallowed for the elements of the second clause. Thus, the following are not WFFs: \( \neg(p), (\neg p), p \lor q \), etc.\(^1\) The upshot is that the definition of a WFF is a primitive ‘syntax’ for our logical language.

This syntax is interesting from a number of perspectives. First, it is extremely simple. Second, it is unambiguous; there is one and only one ‘parse’ for any WFF. Third, it is, in some either really important or really trivial sense, “artificial”. Let’s look at each of these points a little more closely.

### 4.2.1 Simplicity

If we want to compare the structures we have developed with those of natural language, we can see that the structures proposed here are far more limited. The best analogy is that atomic statements are like simple clauses and WFFs are combinations of clauses. Thus, we might see \( p \) as analogous to *Ernie likes logic* and \( q \) as analogous to *Apples grow on trees*. We might then take \( (p \land q) \) as analogous to *Ernie likes logic and apples grow on trees*.

This analogy is fine, but it’s easy to see that the structures allowed by sentential logic are quite primitive in comparison with human language.

First, natural language provides for many many mechanisms to build simple sentences. Sentential logic only allows the “prime”. Second, while we have five ways to build on atomic statements in logic, natural language allows many more ways to combine sentences.

\(^1\)There are some variations in symbology that you’ll find when you look at other texts. The negation symbol in something like \( \neg a \) can also be written \( \sim a \). Likewise, the ‘and’ symbol in something like \( (a \land b) \) can also be written \( (a\&b) \). There are lots of other notational possibilities and parentheses are treated in different ways.
(4.2) Ernie likes logic \hspace{1cm} \text{apples grow on trees.}

\begin{align*}
\text{and} \\
\text{if} \\
\text{so} \\
\text{while} \\
\text{unless} \\
\text{because} \\
\text{though} \\
\text{but} \\
\ldots
\end{align*}

4.2.2 Lack of Ambiguity

On the other hand, this impoverished syntax has a very nice property: it is unambiguous. Recall that sentences in natural language can often be parsed in multiple ways. For example, we saw that a sentence like \textit{Ernie saw the man with the binoculars} has two different meanings associated with two different structures.

This cannot happen with WFFs; structures will always have a single parse. Consider, for example, a WFF like \((p \land (q \land r))\). Here, the fact that \(q\) and \(r\) are grouped together before \(p\) is included is apparent from the parentheses. The other parse would be given by \(((p \land q) \land r)\). An ambiguous structure like \((p \land q \land r)\) is not legal.

4.2.3 Artificiality

Both of these properties limit the use of sentential logic in encoding expressions of human language. We will see below, however, that the semantics of sentential logic are impoverished as well. It turns out that the syntax is demonstrably just powerful enough to express a particular set of meanings.

4.3 Basic Semantics

The set of truth values or meanings that our sentences can have is very impoverished: \(\mathbf{T}\) or \(\mathbf{F}\). We often interpret these as ‘true’ and ‘false’ respectively,
but this does not have to be the case. For example, we might interpret them as ‘blue’ and ‘red’, 0 and 1, ‘apples’ and ‘oranges’, etc. An atomic statement like \( p \) can exhibit either of these values. Likewise, any WFF can bear one of these values.

Let’s go through each of the connectives and see how they affect the meaning of the larger WFF. Negation reverses the truth value of its WFF. If \( p \) is true, then \( \neg p \) is false; if \( p \) is false, then \( \neg p \) is true.

\[
(4.3) \quad \begin{array}{c|c}
p & \neg p \\
T & F \\
F & T \\
\end{array}
\]

Conjunction—logical ‘and’—combines two WFFs. If both are true, the combination is true. In all other cases, the combination is false.

\[
(4.4) \quad \begin{array}{c|c|c}
p & q & (p \land q) \\
T & T & T \\
T & F & F \\
F & T & F \\
F & F & F \\
\end{array}
\]

Disjunction—logical ‘or’—combines two WFFs. If both are false, the combination is false. In all other cases, the combination is true.

\[
(4.5) \quad \begin{array}{c|c|c}
p & q & (p \lor q) \\
T & T & T \\
T & F & T \\
F & T & T \\
F & F & F \\
\end{array}
\]

The conditional—or logical ‘if’—is false just in case the left side is true and the right side is false. In all other cases, it is true.
Finally, the biconditional, or ‘if and only if’, is true when both sides are true or when both sides are false. If the values of the conjuncts do not agree, the biconditional is false.

Let’s do some examples.

- \((p \to q) \lor r\)
- \(\neg(p \leftrightarrow p)\)
- \(\neg\neg\neg(p \lor q)\)
- \((p \lor (q \land (r \lor \neg s)))\)
- \((p \lor \neg p)\)
Let’s consider the first case above: \(((p \rightarrow q) \lor r)\). How do we build a truth table for this? First, we collect the atomic statements: \(p\), \(q\), and \(r\). To compute the possible truth values of the full WFF, we must consider every combination of truth values for the component atomic statements. Since each statement can take on one of two values, there are \(2^n\) combinations for \(n\) statements. In the present case, there are three atomic statements, so there must be \(2^3 = 2 \times 2 \times 2 = 8\) combinations.

There will then be eight rows in our truth table. The number of columns is governed by the number of atomic statements plus the number of instances of the connectives. In the case at hand, we have three atomic statements and two connectives: ‘\(\rightarrow\)’ and ‘\(\lor\)’. Thus there will be five columns in our table.

We begin with columns labeled for the three atomic statements. The rows are populated by every possible combination of their truth values: 8.

\[
\begin{array}{ccc}
 p & q & r \\
 T & T & T \\
 T & T & F \\
 T & F & T \\
 T & F & F \\
 F & T & T \\
 F & T & F \\
 F & F & T \\
 F & F & F \\
\end{array}
\]

We then construct the additional columns by building up from the atomic statements. This effectively means starting from the most embedded WFFs and working outward. In the case at hand, we next construct a column for \((p \rightarrow q)\). We do this from the values in columns one and two, following the pattern outlined in (4.6).
Finally, we construct the last column from the values in columns three and four using the pattern outlined in (4.5).

\[
\begin{array}{c|c|c|c|c|c|c}
 p & q & r & (p \rightarrow q) & (p \rightarrow q) \lor r \\
 T & T & T & T & T \\
 T & T & F & T & T \\
 T & F & T & F & T \\
 T & F & F & F & T \\
 F & T & T & T & T \\
 F & T & F & T & T \\
 F & F & T & T & T \\
 F & F & F & T & T \\
\end{array}
\]

The remaining WFFs are left as an exercise.

### 4.4 The Meanings of the Connectives

The names of the individual connectives and the corresponding methods for constructing truth tables suggest strong parallels with the meanings of various natural language connectives. There are indeed parallels, but it is essential to keep in mind that the connectives of sentential logic have very precise interpretations that can differ wildly from our intuitive understandings of
the corresponding expressions in English. Let’s consider each one of the connectives and see how each differs in interpretation from the corresponding natural language expression.

4.4.1 Negation

The negation connective switches the truth value of the WFF it attaches to. Thus $\neg p$ bears the opposite value from $p$, whatever that is. There are a number of ways this differs from natural language.

First, as we discussed in chapter 2, some languages use two negative words to express a single negative idea. Thus the Spanish Ernie no vio nada ‘Ernie saw nothing’ uses two negative words no and nada to express a single negative. In the version of sentential logic that we have defined, adding a second instance of ‘$\neg$’ undoes the effect of the first. Thus $\neg\neg p$ bears the same truth value as $p$ and the opposite truth value from $\neg p$.

Another difference between natural language negation and formal sentential logic negation can be exemplified with the following pair of sentences:

\begin{align*}
(4.11) & \quad \text{Ernie likes logic.} \\
& \quad \text{Ernie doesn’t like logic.}
\end{align*}

In natural language, these sentences do not exhaust the range of possibilities. Ernie could simply not care. That is, in natural language a sentence and its negation do not exhaust the range of possible meanings. In sentential logic, they do. Thus either $p$ is true or $\neg p$ is true. There is no other possibility.

4.4.2 Conjunction

Natural language conjunction is also different from sentential logic conjunction. Consider, for example, a sentence like the following:

\begin{align*}
(4.12) & \quad \text{Ernie went to the library and Hortence read the book.}
\end{align*}

A sentence like this has several implications beyond whether Ernie went to the library and whether Hortence read some book. In particular, the sentence implies that these events are connected. For example, the book was probably borrowed from the library. Another implication is that the events happened in the order they are given: Ernie first went to the library and then Hortence read the book.
These sorts of connections and implications do not apply to a WFF like \((p \land q)\). All we know is the relationship between the truth value of the whole and the truth values of the parts: \((p \land q)\) is true just in case \(p\) is true and \(q\) is true.

### 4.4.3 Disjunction

Disjunction in natural language is also interpreted differently from logical disjunction. Consider the following example.

\[(4.13)\] Ernie went to the library or Hortence read the book.

A sentence like this has the interpretation that one of the two events holds, but not both. Thus one might interpret this sentence as being true just in case the first part is true and the second false or the second part is true and the first is false, but not if both parts are true.

This is in contrast to a WFF like \((p \lor q)\), which is true if either or both disjuncts are true.

### 4.4.4 Conditional

A natural language conditional is also subject to a different interpretation from the sentential logic conditional. Consider a sentence like the following:

\[(4.14)\] If pigs can fly, Ernie will go to the library.

A sentence like this would normally be interpreted as indicating that Ernie will not be going to the library. That is, if the antecedent is obviously false, the consequent—the second statement—must also be false.

This is not true of the sentential logic conditional. A sentential logic conditional like \((p \to q)\) is false just in case \(p\) is true and \(q\) is false; in all other cases, it is true. Thus, if \(p\) is false, the whole expression is true regardless of the truth value of \(q\).

### 4.4.5 Biconditional

The biconditional is probably the least intuitive of the logical connectives. The expression \(if\ and\ only\ if\) does not appear to be used in colloquial English.
Rather, to communicate the content of a biconditional in spoken English, one frequently hears rather convoluted expressions like the following:

(4.15) Hortence will read the book if Ernie goes to the library and won’t if he doesn’t.

### 4.5 How Many Connectives?

Notice that each of the connectives, except ¬, can be defined in terms of the others. The expressions in the second column below are true just when the expressions in the third column below are true. Thus the connective in the second column can be defined in terms of the connectives in the third column.

\[
\begin{array}{c|c|c|c|c|c}
\text{if} & (p \to q) & (\neg p \lor q) & \text{iff} & (p \leftrightarrow q) & ((p \to q) \land (q \to p)) \\
\text{and} & (p \land q) & \neg (\neg p \lor \neg q) & \text{or} & (p \lor q) & \neg (\neg p \land \neg q) \\
\end{array}
\]

Consider, for example, the truth table below for the first row in (4.16).

\[
\begin{array}{c|c|c|c|c}
p & q & (p \to q) & \neg p & (\neg p \lor q) \\
\hline
T & T & T & F & T \\
T & F & F & F & F \\
F & T & T & T & T \\
F & F & T & T & T \\
\end{array}
\]

Notice that the values for the third and fifth columns are identical, indicating that these expressions have the same truth values under the same conditions. What this means is that any expression with a conditional can be replaced with a negation plus disjunction, and vice versa. The same is true for the other rows in the table in (4.16).

We can, therefore, have a logical system that has the same expressive potential as the one we’ve defined with five connectives, but that has only two connectives: ¬ plus and one of →, ∧, or ∨. For example, let’s choose
\neg \text{ and } \land. \text{ The following chart shows how all three other connectives can be expressed with only combinations of those two connectives.}

\begin{equation}
\text{(4.18) if } \begin{array}{c}
(p \rightarrow q) \\
(p \leftrightarrow q) \\
(p \lor q)
\end{array} \iff \begin{array}{c}
\neg(\neg\neg p \land \neg q) \\
(\neg(\neg\neg p \land \neg q) \land \neg(\neg q \land \neg p)) \\
\neg(\neg p \land \neg q)
\end{array}
\end{equation}

What we’ve done here is take each equivalence in the chart in (4.16) through successive equivalences until all that remains are instances of negation and conjunction. For example, starting with \((p \rightarrow q)\) (first row of the preceding table), we first replace the conditional with negation and disjunction, i.e. \((\neg p \lor q)\). We then use the equivalence of disjunction and conjunction in row three of (4.16) to replace the disjunction with an instance of conjunction, adding more negations in the process. The following truth table for \((p \rightarrow q)\) and \(\neg(\neg p \land \neg q)\) shows that they are equivalent.

\begin{equation}
\text{(4.19) }\begin{array}{c|c|c|c|c|c|c|c|c}
  p & q & (p \rightarrow q) & \neg p & \neg\neg p & \neg q & (\neg p \land \neg q) & \neg(\neg p \land \neg q) \\
  \hline
  T & T & T & F & T & F & F & T \\
  T & F & F & F & T & T & T & F \\
  F & T & T & T & F & F & F & T \\
  F & F & T & T & F & T & F & T
\end{array}
\end{equation}

The third and eighth columns have identical truth values.

In fact, it is possible to have the same expressive potential with only one connective: the Sheffer stroke, e.g. in \((p \mid q)\). A formula built on the Sheffer stroke is false just in case both components are true; else it is true.

\begin{equation}
\text{(4.20) }\begin{array}{c|c|c}
  p & q & (p \mid q) \\
  \hline
  T & T & F \\
  T & F & T \\
  F & T & T \\
  F & F & T
\end{array}
\end{equation}

Consider, for example, how we might use the Sheffer stroke to get negation. The following truth table shows the equivalence of \(\neg p\) and \((p \mid p)\).
The following truth table shows how we can use the Sheffer stroke to get conjunction.

| p | q | (p ∧ q) | (p | q) | ((p | q) | (p | q)) |
|---|---|---------|--------|----------------|
| T | T | T       | F      | T              |
| T | F | F       | T      | F              |
| F | T | F       | T      | F              |
| F | F | F       | T      | F              |

We leave the remaining equivalences as exercises.

### 4.6 Tautology, Contradiction, Contingency

We’ve seen that the truth value of a complex formula can be built up from the truth values of its component atomic statements. Sometimes, this process produces a formula that can be either true or false. Such a formula is referred to as a **contingency**. All the formulas we have given so far have been of this sort. Sometimes, however, the formula can only be true or can only be false. The former is a **tautology** and the latter a **contradiction**.

For example, \( p \rightarrow q \) is a contingency, since it can be true or false.

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>(p → q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

The rightmost column of this truth table contains instances of T and instances of F. Notice that there are no “degrees” of contingency. If both values are possible, the formula is contingent.
We also have formulas that can only be true: tautologies. For example, \( p \rightarrow p \) is a tautology.

\[
\begin{array}{c|c}
 p & (p \rightarrow p) \\
\hline
 T & T \\
 F & T \\
\end{array}
\]

Even though \( p \) can be either true or false, the combination can only bear the value \( T \). A similar tautology can be built using the biconditional.

\[
\begin{array}{c|c}
 p & (p \leftrightarrow p) \\
\hline
 T & T \\
 F & T \\
\end{array}
\]

Likewise, we have contradictions: formulas that can only be false, e.g. \((p \land \neg p)\).

\[
\begin{array}{c|c|c}
 p & \neg p & (p \land \neg p) \\
\hline
 T & F & F \\
 F & T & F \\
\end{array}
\]

Again, even though \( p \) can have either value, this combination of connectives can only be false.

### 4.7 Proof

We can now consider proof and argument. Our goal is to show how we can reason from some set of WFFs to some specific conclusion, e.g. another WFF. We’ll start with the simplest cases: reasoning from one WFF to another or reasoning that some WFF is a tautology.

Notice first that tautologies built on the biconditional entail that the two sides of the biconditional are logical equivalents. That is, each side is true just in case the other side is true and each side is false just in case the other side is false. For example, \(((p \land q) \leftrightarrow \neg (\neg p \lor \neg q))\) has this property. \((p \land q)\)
is logically equivalent to \( \neg(\neg p \lor \neg q) \). If we did truth tables for these two WFFs, we would see that they bear the same truth values.

Conditionals are similar, though not as powerful. For example: \( ((p \land q) \rightarrow p) \). The atomic statement \( p \) is a logical consequence of \( p \land q \), and is true whenever the latter is true. However, the converse is not the case; \( p \land q \) does not follow from \( p \). We can see this in the following truth table.

\[
\begin{array}{ccc|c}
\text{p} & \text{q} & (p \land q) & ((p \land q) \rightarrow p) \\
T & T & T & T \\
T & F & F & T \\
F & T & F & T \\
F & F & F & T \\
\end{array}
\]

The rightmost column has only instances of \( T \), so this WFF is a tautology. However, comparison of the second and third columns shows that the two sides of the formula do not bear the same value in all instances; they are not logical equivalents. In particular, row two of the table shows that the left side can be false when the right side is true. This, of course, is allowed by a conditional, as opposed to a biconditional, but it exemplifies directly that the values of the two sides need not be identical for the whole WFF to be a tautology.

We can use these relationships—along with a general notion of substitution—to build arguments. The basic idea behind substitution is that if some formula \( A \) contains \( B \) as a subformula, and \( B \) is logically equivalent to \( C \), then \( C \) can be substituted for \( B \) in \( A \). For example, we can show by truth table that \( p \leftrightarrow (p \land p) \).

\[
\begin{array}{c|c}
p & (p \land p) \\
T & T \\
F & F \\
\end{array}
\]

It therefore follows that \( p \) and \( (p \land p) \) are logical equivalents. Hence, any formula containing \( p \) can have all instances of \( p \) replaced with \( (p \land p) \) without altering its truth value. For example, \( (p \rightarrow q) \) has the same truth value as \( ((p \land p) \rightarrow q) \). This is shown in the following truth table.
The values for the fourth and fifth columns are the same.

Substitution for logical consequence is more restricted. If some formula $A$ contains $B$ as a subformula, and $C$ is a logical consequence of $B$, then $C$ cannot be substituted for $B$ in $A$. A logical consequence can only be substituted for the entire formula. Thus if $B$ appears by itself, $C$ may be substituted for it.

Consider this example: $(p \land \neg p) \rightarrow q)$. First, we can show that this is a tautology by inspection of the truth table.

The rightmost column contains only instances of $T$; hence the WFF is a tautology. The last connective is a conditional and the values of the two sides do not match in all cases—in particular rows one and three; hence the second half of the formula $q$ is a logical consequence of the first $(p \land \neg p)$.

Now, by the same reasoning $((p \land \neg p) \rightarrow r)$ is also a tautology. However, if we erroneously do a partial substitution on this second formula, substituting $q$ for the antecedent $(p \land \neg p)$ based on the first formula, the resulting formula $(q \rightarrow r)$ is certainly not a tautology.
Hence, though partial substitution based on logical equivalence preserves truth values, partial substitution based on logical consequence does not.

There are a number of logical equivalences and consequences that can be used for substitution. We will refer to these as the *Laws of Sentential Logic*. The most interesting thing to note about them is that they are very much like the laws for set theory in section 3.4. In all of the following we use the symbol ‘$\iff$’ to indicate explicitly that the subformulas to each side can be substituted for each other in any WFF.

Idempotency says that the disjunction or conjunction of identical WFFs is identical to the WFF itself. This can be compared with set-theoretic idempotency in (3.9).

\[(4.32)\quad \text{Idempotency} \]
\[
\begin{align*}
(p \lor p) & \iff p \\
(p \land p) & \iff p
\end{align*}
\]

Associativity says that the order in which successive conjunctions or successive disjunctions are applied can be switched. This can be compared with (3.11).

\[(4.33)\quad \text{Associativity} \]
\[
\begin{align*}
((p \lor q) \lor r) & \iff (p \lor (q \lor r)) \\
((p \land q) \land r) & \iff (p \land (q \land r))
\end{align*}
\]

Commutativity says that the order of conjuncts and disjuncts is irrelevant. Compare with (3.10).
(4.34) Commutativity
\[(p \lor q) \iff (q \lor p)\]
\[(p \land q) \iff (q \land p)\]

Distributivity can "lower" a conjunction into a disjunction or lower a disjunction into a conjunction. This can be compared with (3.12).

(4.35) Distributivity
\[(p \lor (q \land r)) \iff ((p \lor q) \land (p \lor r))\]
\[(p \land (q \lor r)) \iff ((p \land q) \lor (p \land r))\]

The truth value $T$ is the identity element for conjunction and the value $F$ is the identity element for disjunction. This is analogous to the roles of $\emptyset$ and $U$ in union and intersection, respectively: (3.13).

(4.36) Identity
\[(p \lor F) \iff p\]
\[(p \land T) \iff p\]

We have an equivalent to set-theoretic Domination (3.14) as well.

(4.37) Domination
\[(p \lor T) \iff T\]
\[(p \land F) \iff F\]

The Complement Laws govern the role of negation and can be compared with (3.15).

(4.38) Complement Laws
\[(p \lor \neg p) \iff T\]
\[(p \land \neg p) \iff F\]
There is also an equivalent to the set-theoretic Double Complement Law (3.16) that governs multiple negations.

(4.39) Double Complement Law
\[ \neg \neg p \iff p \]

DeMorgan’s Laws allow us to use negation to relate conjunction and disjunction, just as set complement can be used to relate intersection and union: (3.18).

(4.40) DeMorgan’s Laws
\[ \neg (p \lor q) \iff (\neg p \land \neg q) \]
\[ \neg (p \land q) \iff (\neg p \lor \neg q) \]

The Conditional and Biconditional Laws define conditionals and biconditionals in terms of the other connectives.

(4.41) Conditional Laws
\[ (p \rightarrow q) \iff (\neg p \lor q) \]
\[ (p \rightarrow q) \iff (\neg q \rightarrow \neg p) \]
\[ (p \rightarrow q) \iff \neg (p \land \neg q) \]

(4.42) Biconditional Laws
\[ (p \iff q) \iff ((p \rightarrow q) \land (q \rightarrow p)) \]
\[ (p \iff q) \iff ((\neg p \land \neg q) \lor (p \land q)) \]

We can build arguments with these. For example, we can use these to show that this statement is a tautology: \(((p \rightarrow q) \land p) \rightarrow q\). First, we know this is true because of the truth table.

\[
\begin{array}{c|c|c|c|c}
p & q & (p \rightarrow q) & ((p \rightarrow q) \land p) & (((p \rightarrow q) \land p) \rightarrow q) \\
\hline
T & T & T & T & T \\
T & F & F & F & T \\
F & T & T & F & T \\
F & F & T & F & T \\
\end{array}
\]
We can also show that it’s a tautology using the Laws of Sentential Logic above. We start out by writing the WFF we are interested in. We then invoke the different laws one by one until we reach a $T$.

(4.44) 1 $(((p \rightarrow q) \land p) \rightarrow q)$ Given
2 $(((\neg p \lor q) \land p) \rightarrow q)$ Conditional
3 $(\neg((\neg p \lor q) \land p) \lor q)$ Conditional
4 $((\neg(\neg p \lor q) \lor \neg p) \lor q)$ DeMorgan
5 $((\neg(\neg p \lor q) \lor (\neg p \lor q))$ Associativity
6 $((\neg p \lor q) \lor (\neg(\neg p \lor q))$ Commutativity
7 $T$ Complement

The first two steps eliminate the conditionals using the Conditional Laws. We then use DeMorgan’s Law to replace the conjunction with a disjunction. We rebracket the expression using Associativity and reorder the terms using Commutativity. Finally, we use the Complement Laws to convert the WFF to $T$.

Here’s a second example: $(p \rightarrow (\neg q \lor p))$. We can show that this is tautologous by truth table:

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\neg q$</th>
<th>$(\neg q \lor p)$</th>
<th>$(p \rightarrow (\neg q \lor p))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

We can also show this using substitution and the equivalences above.

(4.46) 1 $(p \rightarrow (\neg q \lor p))$ Given
2 $(\neg p \lor (\neg q \lor p))$ Conditional
3 $((\neg q \lor p) \lor \neg p)$ Commutativity
4 $(\neg q \lor (p \lor \neg p))$ Associativity
5 $(\neg q \lor T)$ Complement
6 $T$ Domination
First, we remove the conditional with the Conditional Laws and reorder the disjuncts with Commutativity. We then rebracket with Associativity. We use the Complement Laws to replace \((p \lor \neg p)\) with \(T\), and Domination to pare this down to \(T\).

So far, we have done proofs that some particular WFF is a tautology, where we reason from some WFF to \(T\). We can also do proofs of one formula from another. In this case, the first line of the proof is the first formula and the last line of the proof must be the second formula. For example, we can prove \((p \land q)\) from \(\neg(\neg q \lor \neg p)\) as follows.

\[(4.47)\]

\[
\begin{array}{cccc}
1 & \neg(\neg q \lor \neg p) & \text{Given} \\
2 & (\neg
\neg q \land \neg
\neg p) & \text{DeMorgan} \\
3 & (q \land \neg
\neg p) & \text{Doub. Comp.} \\
4 & (q \land p) & \text{Doub. Comp.} \\
5 & (p \land q) & \text{Commutativity} \\
\end{array}
\]

First we use DeMorgan’s Law to eliminate the disjunction. We then use the Double Complement Law twice to eliminate the negations and Commutativity to reverse the order of the terms.

We interpret such a proof as indicating that the the ending WFF is true if the starting WFF is true, in this case, that if \(\neg(\neg q \lor \neg p)\) is true, \((p \land q)\) is true. We will see in section 4.9 that it follows that \((\neg(\neg q \lor \neg p) \rightarrow (p \land q))\) must be tautologous.

Finally, let’s consider an example where we need to use a logical consequence, rather than a logical equivalence. First, note that \(((p \land q) \rightarrow p)\) has \(p\) as a logical consequence of \((p \land q)\). That is, if we establish that \(((p \land q) \rightarrow p)\), we will have that \(p\) can be substituted for \((p \land q)\). We use ‘\(\Rightarrow\)’ to denote that a term can be substituted by logical consequence: \((p \land q) \Rightarrow p\). We can show this by truth table:

\[(4.48)\]

\[
\begin{array}{ccc|c}
\hline
p & q & (p \land q) & ((p \land q) \rightarrow p) \\
\hline
T & T & T & T \\
T & F & F & T \\
F & T & F & T \\
F & F & F & T \\
\hline
\end{array}
\]
Whenever T appears in the third column, T appears in the fourth column. It now follows that \((p \land q) \implies p\).

Incidentally, note that the following truth table shows that the biconditional version of this WFF, \(((p \land q) \iff p)\), is not a tautology.

\[
\begin{array}{ccc|c}
 p & q & (p \land q) & ((p \land q) \iff p) \\
\hline
 T & T & T & T \\
 T & F & F & F \\
 F & T & F & T \\
 F & F & F & T \\
\end{array}
\]

This is not a tautology because there is an instance of F in the rightmost column. We now have: \((p \land q) \implies p\). We can use this—in conjunction with the Laws of Sentential Logic—to prove \(q\) from \(((p \rightarrow q) \land p)\).

\[
\begin{align*}
(4.49) & \quad \begin{array}{c|c|c|c}
 p & q & (p \land q) & ((p \land q) \iff p) \\
\hline
 T & T & T & T \\
 T & F & F & F \\
 F & T & F & T \\
 F & F & F & T \\
\end{array} \\
\text{1} & \quad ((p \rightarrow q) \land p) & \text{Given} \\
\text{2} & \quad ((\neg p \lor q) \land p) & \text{Conditional} \\
\text{3} & \quad ((\neg p \land p) \lor (q \land p)) & \text{Distributivity} \\
\text{4} & \quad ((p \land \neg p) \lor (q \land p)) & \text{Commutativity} \\
\text{5} & \quad (F \lor (q \land p)) & \text{Complement} \\
\text{6} & \quad ((q \land p) \lor F) & \text{Commutativity} \\
\text{7} & \quad (q \land p) & \text{Identity} \\
\text{8} & \quad q & \text{Established just above}
\end{align*}
\]

First, we remove the conditional with the Conditional Laws. We lower the conjunction using Distributivity and reorder terms with Commutativity. We use the Complement Laws to replace \((p \land \neg p)\) with F and reorder again with Commutativity. We now remove the F with Identity and use our newly proven logical consequence to replace \((q \land p)\) with q.

Summarizing thus far, we’ve outlined the basic syntax and semantics of propositional logic. We’ve shown how WFFs can be categorized into contingencies, tautologies, and contradictions. We’ve also shown how certain WFFs are logical equivalents and others are logical consequences. We’ve used the Laws of Sentential Logic to construct simple proofs.
4.8 Rules of Inference

The Laws of Sentential Logic allow us to progress in a proof from one WFF to another, resulting in a WFF or truth value at the end. For example, we now know that we can prove $q$ from $((p \rightarrow q) \land p)$. We can also show that $(((p \rightarrow q) \land p) \rightarrow q)$ is a tautology.

The Laws do not allow us to reason over sets of WFFs, however. The problem is that they do not provide a mechanism to combine the effect of separate WFFs. For example, we do not have a mechanism to reason from $(p \rightarrow q)$ and $p$ to $q$.

There are Rules of Inference that allow us to do this. We now present some of the more commonly used ones.

*Modus Ponens* allows us to deduce the consequent of a conditional from the conditional itself and its antecedent.

\[(4.51) \text{ Modus Ponens } \quad P \rightarrow Q \]
\[
\begin{array}{c}
\text{M.P.} \\
\hline \hline
P \\
Q
\end{array}
\]

*Modus Tollens* allows us to conclude the negation of the antecedent from a conditional and the negation of its consequent.

\[(4.52) \text{ Modus Tollens } \quad P \rightarrow Q \]
\[
\begin{array}{c}
\text{M.T.} \\
\hline \hline
\neg Q \\
\neg P
\end{array}
\]

Hypothetical Syllogism allows us to use conditionals transitively.

\[(4.53) \text{ Hypothetical Syllogism } \quad P \rightarrow Q \]
\[
\begin{array}{c}
\text{H.S.} \\
\hline \hline
Q \rightarrow R \\
P \rightarrow R
\end{array}
\]

Disjunctive Syllogism allows us to conclude the second disjunct from a disjunction and the negation of its first disjunct.
Chapter 4. Sentential Logic

(4.54) Disjunctive Syllogism \[ P \lor Q \]
D.S. \[ \neg P \]
\[ \frac{Q}{Q} \]

Simplification allows us to conclude that the first conjunct of a conjunction must be true.\(^2\) Note that this is a logical consequence and not a logical equivalence.

(4.55) Simplification \[ P \land Q \]
Simp. \[ \frac{P}{P} \]

Conjunction allows us to assemble two independent WFFs into a conjunction.\(^3\)

(4.56) Conjunction \[ P \]
Conj. \[ Q \]
\[ \frac{P \land Q}{P \land Q} \]

Finally, Addition allows us to disjoin a WFF and anything.\(^4\)

(4.57) Addition \[ P \]
Add. \[ \frac{P \lor Q}{P \lor Q} \]

Note that some of the Rules have funky Latin names and corresponding abbreviations. These are not terribly useful, but we’ll keep them for conformity with other treatments.

Let’s consider a very simple example. Imagine we are trying to prove \( q \) from \(( p \rightarrow q)\) and \( p \). We can reason from the two premises to the conclusion with \textit{Modus Ponens} (M.P.). The proof is annotated as follows.

\(^2\)This is also called \textit{Conjunction Elimination}.
\(^3\)This is also called \textit{Adjunction} or \textit{Conjunction Introduction}.
\(^4\)This is also called \textit{Disjunction Introduction}.
(4.58) 1 \((p \to q)\) Given
2 \(p\) Given
3 \(q\) 1,2 M.P.

There are two formulas given at the beginning. We then use them and Modus Ponens to derive the third step. Notice that we explicitly give the line numbers that we are using Modus Ponens on.

Here’s a more complex example. We want to prove \(p\) from \((t \to q)\), \((t \lor s)\), \((q \to r)\), \((s \to p)\), and \(\neg r\).

(4.59) 1 \((t \to q)\) Given
2 \((t \lor s)\) Given
3 \((q \to r)\) Given
4 \((s \to p)\) Given
5 \(\neg r\) Given
6 \(\neg q\) 3,5 M.T.
7 \(\neg t\) 1,6 M.T.
8 \(s\) 2,7 D.S.
9 \(p\) 4,8 M.P.

First, we give the five statements that we are starting with. We then use Modus Tollens on the third and fifth to get \(\neg q\). We use Modus Tollens again on the first and sixth to get \(\neg t\). We can now use Disjunctive Syllogism to get \(s\) and Modus Ponens to get \(p\), as desired.

Notice that the Laws and Rules are redundant. For example, we use Modus Tollens above to go from steps 3 and 5 to step 6. We could instead use one of the Conditional Laws to convert \((q \to r)\) to \((\neg r \to \neg q)\), and then use Modus Ponens on that and \(\neg r\) to make the same step. That is, Modus Tollens is unnecessary if we already have Modus Ponens and the Conditional Laws.

Here’s another example showing how the Laws of Sentential Logic can be used to massage WFFs into a form that the Rules of Inference can be applied to. Here we wish to prove \((p \to r)\) from \((p \to (q \lor r))\) and \(\neg q\).
(4.60) 1  \((p \to (q \lor r))\) Given
2  \(\neg q\) Given
3  \((\neg p \lor (q \lor r))\) 1 Cond.
4  \(((q \lor r) \lor \neg p)\) 3 Comm.
5  \((q \lor (r \lor \neg p))\) 4 Assoc.
6  \((r \lor \neg p)\) 2,5 D.S.
7  \((\neg p \lor r)\) 6 Comm.
8  \((p \to r)\) 7 Cond.

As usual, we first eliminate the conditional with the Conditional Laws. We then reorder with Commutativity and rebracket with Associativity. We use Disjunctive Syllogism on lines 2 and 5, reorder, and then reinsert the conditional.

### 4.9 Conditional Proof

A special method of proof is available when you want to prove a conditional. Imagine you have a conditional of the form \((A \to B)\) that you want to prove. What you do is assume \(A\) and attempt to prove \(B\) from it. If you can, then you can conclude \((A \to B)\).

Here’s an example showing how to prove \((p \to r)\) from \((p \to (q \lor r))\) and \(\neg q\).\

(4.61) 1  \((p \to (q \lor r))\) Given
2  \(\neg q\) Given
3  \(p\) Auxiliary Premise
4  \((q \lor r)\) 1,3 M.P.
5  \(r\) 2,4 D.S.
6  \((p \to r)\) 3–5 Conditional Proof

We indicate a Conditional Proof with a left bar. The bar extends from the assumption of the antecedent of the conditional—the “Auxiliary Premise”—to the point where we invoke Conditional Proof. The key thing to keep in mind is that we assume \(A\) and try to prove \(B\) from it.

---

\(^5\)We just proved this above directly on page 71.
mind is that we cannot refer to anything to the right of that bar below the end of the bar. In this case, we are only assuming \( p \) to see if we can get \( r \) to follow. We cannot conclude on that basis that \( p \) is true on its own.

We can show that this is the case with a fallacious proof of \( p \) from no assumptions. First, here’s an example of Conditional Proof used correctly: we prove \((p \rightarrow p)\) from no assumptions.

\[
\begin{array}{c|ccc}
(4.62) & 1 & p & \text{Auxiliary Premise} \\
      & 2 & p & \text{1 Repeated} \\
      & 3 & (p \rightarrow p) & 1–2 \text{ Conditional Proof} \\
\end{array}
\]

Now we add an incorrect step, referring to material to the right of the bar below the end of the bar.

\[
\begin{array}{c|ccc}
(4.63) & 1 & p & \text{Auxiliary Premise} \\
      & 2 & p & \text{1 Repeated} \\
      & 3 & (p \rightarrow p) & 1–2 \text{ Conditional Proof} \\
      & 4 & p & \text{1 Repeated Wrong!} \\
\end{array}
\]

Intuitively, this should be clear. The proof above does not establish the truth of \( p \). Here’s another very simple example of Conditional Proof. We prove \(((p \rightarrow q) \rightarrow (\neg p \lor q))\).

\[
\begin{array}{c|ccc}
(4.64) & 1 & (p \rightarrow q) & \text{Auxiliary Premise} \\
      & 2 & (\neg p \lor q) & 1 \text{ Cond.} \\
      & 3 & ((p \rightarrow q) \rightarrow (\neg p \lor q)) & 1–2 \text{ Conditional Proof} \\
\end{array}
\]

The important thing is to understand the intuition behind Conditional Proof. If you want to show that \( p \) follows from \( q \), assume that \( q \) is true and see if you get \( p \). Let’s use it now to prove \((p \rightarrow q)\) from \( q \).

\[
\begin{array}{c|ccc}
(4.65) & 1 & q & \text{Given} \\
      & 2 & p & \text{Auxiliary Premise} \\
      & 3 & q & 1, \text{ Repeated} \\
      & 4 & (p \rightarrow q) & 2–3 \text{ Conditional Proof} \\
\end{array}
\]

Notice in this last proof that when we are to the right of the bar we can refer to something above and to the left of the bar.
4.10 Indirect Proof

A proof technique with a similar structure to Conditional Proof that has wide applicability is *Indirect Proof* or *Reductio ad Absurdum*. Imagine we are attempting to prove \( q \). We assume \( \neg q \) and try to prove a contradiction from it. If we succeed, we have proven \( q \). The reasoning is like this. If we can prove a contradiction from \( \neg q \), then there’s no way \( \neg q \) can be true. If it’s not true, it *must* be false. Here’s a simple example, where we attempt to prove \( (p \lor q) \) from \( (p \land q) \).

\[
\begin{align*}
(4.66) \quad 1 & \quad (p \land q) \quad \text{Given} \\
2 & \quad \neg(p \lor q) \quad \text{Auxiliary Premise} \\
3 & \quad (\neg p \land \neg q) \quad 2 \text{ DeMorgan} \\
4 & \quad p \quad 1 \text{ Simp.} \\
5 & \quad \neg p \quad 3 \text{ Simp.} \\
6 & \quad (p \land \neg p) \quad 4,5 \text{ Conj.} \\
7 & \quad (p \lor q) \quad 2–6 \text{ Indirect Proof}
\end{align*}
\]

The indirect portion of the proof is notated like Conditional Proof. Moreover, as with Conditional Proof, the material to the right of the bar cannot be referred to below the bar. Notice too that when we are in the ‘indirect’ portion of the proof, we can refer to material above the bar and to the left. The restriction on referring to material to the right of the bar is therefore “one-way”, just as with Conditional Proof.

Here’s another example of Indirect Proof. We prove \( \neg r \) from \( p \) and \( (r \rightarrow \neg p) \). First, here is one way to prove this via a direct proof.

\[
\begin{align*}
(4.67) \quad 1 & \quad p \quad \text{Given} \\
2 & \quad (r \rightarrow \neg p) \quad \text{Given} \\
3 & \quad (\neg \neg p \rightarrow \neg r) \quad 2 \text{ Cond.} \\
4 & \quad (p \rightarrow \neg r) \quad 3 \text{ Doub. Comp.} \\
5 & \quad \neg r \quad 1,4 \text{ M.P.}
\end{align*}
\]

The proof begins as usual with the two assumptions. We reverse the conditional with the Conditional Laws and remove the double negation with the Double Complement Law. Finally, we use *Modus Ponens* to get \( \neg r \).

\[\text{Note that this argument goes through only if there are only two values to our logic!}\]
Following, we prove the same thing using Indirect Proof.

\[(4.68)\]

<table>
<thead>
<tr>
<th>Step</th>
<th>Formula</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( p )</td>
<td>Given</td>
</tr>
<tr>
<td>2</td>
<td>((r \to \neg p))</td>
<td>Given</td>
</tr>
<tr>
<td>3</td>
<td>(\neg r)</td>
<td>Auxiliary Premise</td>
</tr>
<tr>
<td>4</td>
<td>( r )</td>
<td>3 Doub. Comp.</td>
</tr>
<tr>
<td>5</td>
<td>(\neg p)</td>
<td>2,4 M.P.</td>
</tr>
<tr>
<td>6</td>
<td>((p \land \neg p))</td>
<td>1,5 Conj.</td>
</tr>
<tr>
<td>7</td>
<td>(\neg r)</td>
<td>3–6 Indirect Proof</td>
</tr>
</tbody>
</table>

We begin with the same two assumptions. Since we are interested in proving \(\neg r\), we assume the negation of that, \(\neg\neg r\), and attempt to prove a contradiction from it. First, we remove the double negation with the Double Complement Laws and use *Modus Ponens* to extract \(\neg p\). This directly contradicts our first assumption. We conjoin these and that contradiction completes the Indirect Proof, allowing us to conclude \(\neg r\).

**4.11 Language**

There are three ways in which a system like this is relevant to language.

First, we can view sentence logic as a primitive language with a very precise syntax and semantics. The system differs from human language in several respects, but it can be seen as an interesting testbed for hypotheses about syntax, semantics, and the relation between them.

Second, we can use the theory of proof as a model for grammatical description. In fact, this is arguably the basis for modern *generative grammar*.

We’ve set up a system where we can use proofs to establish that some statement is or is not a tautology. For example, we can show that something like \(\neg(p \land \neg p)\) is tautologous as follows.

\[(4.69)\]

<table>
<thead>
<tr>
<th>Step</th>
<th>Formula</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(\neg(p \land \neg p))</td>
<td>Given</td>
</tr>
<tr>
<td>2</td>
<td>((\neg p \lor \neg \neg p))</td>
<td>1 DeMorgan</td>
</tr>
<tr>
<td>3</td>
<td>((\neg p \lor p))</td>
<td>2 Doub. Compl.</td>
</tr>
<tr>
<td>4</td>
<td>((p \lor \neg p))</td>
<td>3 Commut.</td>
</tr>
<tr>
<td>5</td>
<td>(T)</td>
<td>4 Compl.</td>
</tr>
</tbody>
</table>
The steps are familiar from preceding proofs. The key intuition, however, is that these steps demonstrate that our Laws and Rules of Inference will transform the assumptions that the proof begins with into the conclusion at the end. We will see later that a similar series of steps can be used to show that some particular sentence is well-formed in a language with respect to a grammatical description.

4.12 Summary

This chapter has introduced the basic syntax and semantics of sentential logic. We provided a simple syntax and semantics for atomic statements and the logical connectives that can build on them. The system was very simple, but has the virtue of being unambiguous syntactically. In addition, the procedure for computing the truth value of a complex WFF is quite straightforward.

We also introduced the notion of truth table. These can be used to compute the truth value of a complex WFF. We saw that there are three basic kinds of WFFs: tautologies, contradictions, and contingencies. Tautologies can only be true; contradictions can only be false; and contingencies can be either.

We examined the logical connectives more closely. We saw that, while they are similar in meaning to some human language constructions, they are not quite the same. In addition, we saw that we don’t actually need as many connectives as we have. In fact, we could do with a single connective, e.g. the Sheffer stroke.

We then turned to proof, showing how we could reason from one WFF to another and that this provided a mechanism for establishing that some particular WFF is tautologous. We proposed sets of Laws and Rules of Inference to justify the steps in proofs.

Last, we considered two special proof techniques: Conditional Proof and Indirect Proof. We use Conditional Proof to establish that some conditional expression is tautologous. We use Indirect Proof to establish that the negation of some WFF is tautologous.
4.13 Exercises

1. Construct truth tables for the remaining WFFs on page 51.

2. Express \((p \rightarrow q)\), \((p \leftrightarrow q)\), and \((p \lor q)\) using only the Sheffer stroke.

3. Give a contradictory WFF, using different techniques from those exemplified in the chapter.

4. Give a tautologous WFF, using different techniques from those exemplified in the chapter.

5. Prove \(\neg r\) from \((r \rightarrow q)\), \((q \rightarrow p)\) and \(\neg p\).

6. Prove \(r\) from \(p\), \(\neg q\), and \(((\neg q \land p) \rightarrow r)\).

7. Prove \((p \lor q)\) from \((p \land q)\) using Conditional Proof.

8. Prove \((p \rightarrow s)\) from \((p \rightarrow \neg q)\), \((r \rightarrow q)\), and \((\neg r \rightarrow s)\).

9. There are other ways than the Sheffer stroke to reduce the connectives to a single connective. Can you work out another that would work and show that it does? \(\text{(This is difficult)}\).

10. Prove \(r\) from \((p \land \neg p)\).

11. Prove \(\neg r\) from \((p \land \neg p)\).

12. Explain your answers to the two preceding questions.

13. Prove \((p \rightarrow q)\) from \(q\) using direct proof, indirect proof, and conditional proof.

14. Find rules of grammar that can be expressed with each of the logical connectives.

15. Can you use the proof techniques we’ve developed in this chapter to prove anything useful about the rules you found in the preceding question?
Chapter 5

Predicate Logic

In this chapter, we consider predicate logic, with functions and quantifiers. The discussion is broken up into syntax, semantics, and proofs.

Predicate logic is a richer system than sentential logic and allows us to move closer to the kind of system we need for natural language semantics. It allows us to introduce model theory at an introductory level, a set-theoretic characterization of predicate logic semantics.

5.1 Syntax

The syntax of predicate logic can be broken up into a vocabulary and a set of rules for constructing formulas out of that vocabulary. Basically, the primitives are constants, variables, predicates, connectives, quantifiers, and delimiters. Constants and variables correspond to objects in our world. Constants are like names, e.g. Ernie or Hortence. Variables are more like pronouns, e.g. they or it. Predicates allow us to describe properties of objects and sets of objects. Quantifiers allow us to refer to sets of things. The connectives and delimiters are the same ones from sentential logic.

These are the elements of the vocabulary of first-order predicate logic:

**Constants** $a, b, c, \ldots$; with or without primes. The primes allow us to convert a finite set of letters into an infinite set of constants.

**Variables** $x, y, z, \ldots$; with or without primes. Again, the primes allow us to convert a finite set of letters into an infinite set of variables.

---

1We do not consider identity or function terms.
Terms Constants $\cup$ Variables = Terms. That is, all the constants and variables grouped together constitute the terms of predicate logic.

Predicates $F, G, H, \ldots$. Each takes a fixed number of terms. These too can be marked with primes to get an infinite set of predicates.

Connectives The usual suspects: $\land, \lor, \neg, \rightarrow, \leftrightarrow$. We use the same connectives as in sentential logic.

Quantifiers $\forall$ and $\exists$. The first is the universal quantifier and the second is the existential quantifier.

Delimiters Parentheses.

With primes, there are an infinite number of constants, variables, and predicates.

The vocabulary is used to define the set of WFFs recursively.

1. If $P$ is an $n$-ary predicate and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is a WFF. (Notice that the number of terms present must match the number of terms required by the predicate.)

2. If $\varphi$ and $\psi$ are WFFs, then $\neg \varphi$, $(\varphi \land \psi)$, $(\varphi \lor \psi)$, $(\varphi \rightarrow \psi)$, and $(\varphi \leftrightarrow \psi)$ are WFFs.

3. If $\varphi$ is a WFF and $x$ is a variable, then $(\forall x) \varphi$ and $(\exists x) \varphi$ are WFFs. (The scope of the quantifiers in these WFFs is $\varphi$.)

4. That’s it.

The rules here are straightforward, but there are nuances you need to be alert to.

Notice that there is no syntactic requirement that the variable next to the quantifier be paired with the same variable in the formula in its scope. Thus $(\exists x)F(x)$ is as well-formed a formula as $(\exists x)F(y)$. In fact, $(\exists x)(\forall y)(\forall y)F(z)$ is just as well-formed.

Notice too that we’ve defined a notion scope. For example, in a WFF like $(\forall x)(\forall y)G(x, y)$, the scope of $(\forall y)$ is $G(x, y)$ and the scope of $(\forall x)$ is $(\forall y)G(x, y)$. Notice too that the scope of a quantifier is not simply everything to the right. In a WFF like $((\exists x)G(y) \land F(a))$, the scope of $(\exists x)$ is only $G(y)$. 
On the other hand, in a WFF like \( (\forall x)(F(a) \land G(b)) \), the scope of \( \forall x \) is \( (F(a) \land G(b)) \). Notice how important parentheses and the syntax of a WFF are to determining scope. We will make use of scope in the semantics of predicate logic.

## 5.2 Semantics

The semantics of predicate logic can be understood in terms of set theory. First, we define the notion of *model*.

**Definition 6 (Model)** A model is a set \( D \) and a function \( f \).

1. \( f \) assigns each constant to a member of \( D \).
2. \( f \) assigns each one-place predicate to a subset of \( D \).
3. \( f \) assigns each two-place predicate to a subset of \( D \times D \).
4. etc.

The basic idea is that a set of constants and predicates are paired with elements from the set of elements provided by the model. We can think of those elements as things in the world. Each constant can be paired directly with an element of the model. Thus if our model includes the individual *Ernie*, we might pair the constant \( a \) with that individual, e.g. \( f(a) = \text{Ernie} \in D \).

Predicates are a little more complex. We can think of each predicate as holding either for a set of individual elements of the model or a set of ordered tuples defined in the model. Each predicate thus defines a set of elements or tuples of elements. Thus, if the predicate \( G \) is taken to define the set of individuals that might be in the kitchen at some particular time, we might take \( G \) to be defined as follows: \( f(G) = \{ \text{Ernie, Hortence} \} \subseteq D \).

Predicates with more than one argument are mapped to a set of tuples. For example, if our model is the set of sounds of English and \( F \) is defined as the set of consonants in English that are paired for voicing, we would have \( f(F) = \{ \langle p, b \rangle, \langle t, d \rangle, \langle k, g \rangle, \ldots \} \).

Voicing refers to vibration of the vocal folds. The difference can be felt if you say the sounds in isolation with your finger placed gently to the side of your Adam’s apple: there is vibration with the voiced sound during the consonant.
Notice that there is no requirement that there be a single model. A logical system can be paired with any number of models. Thus a predicate $G$ could be paired with a model of individuals and rooms and define a set of individuals that are in the kitchen or it could be paired instead with a model of sounds in English and define the set of nasal consonants.

We can use model theory to understand how predicate logic formulas are evaluated. As with simple sentential logic, predicate logic formulas evaluate to one of two values: $T$ or $F$. The logical connectives have their usual function, but we must now include a mechanism for understanding predicates and quantifiers.

Consider predicates first. An expression like $G(a)$ is true just in case $f(a)$ is in the subset of $D$ that $f$ assigns $G$ to. For example, if $a$ is paired with Ernie and Ernie is in the set of $D$ that $G$ is paired with, $F(a) = \text{Ernie}$ and Ernie $\in f(G)$, then $G(a)$ is true.

Likewise, $H(a, b)$ is true just in case $\langle a, b \rangle$ is in the subset of $D \times D$ that $f$ assigns $H$ to. For example, if we take $D$ to be the set of words of English and we take $H$ to be the relation 'has fewer letters than', then $H(a, b)$ is true just in case the elements we pair $a$ and $b$ with are in the set of ordered pairs defined by $f(H)$. For example, if $f(a) = \text{hat}$ and $f(b) = \text{chair}$, then $\langle \text{hat}, \text{chair} \rangle \in f(H)$ and $H(a, b)$ is true.

Quantified variables are then understood as follows. Variables range in value over the members of $D$. An expression quantified with $\forall$ is true just in case the expression is true of all members of the model $D$; an expression quantified with $\exists$ is true just in case the expression is true of at least one member of the model $D$. Let’s go through some examples of this. Assume for these examples, that $D = \{m, n, η\}$.

For example, an expression like $(\forall x)G(x)$ is true just in case every member of $D$ is in the set $f$ assigns to $G$. In the case at hand, we might think of $G$ as 'is nasal’, in which case, $f(G) = \{m, n, η\}$. Since $f(G) = D$, $(\forall x)G(x)$ is true. On the other hand, if we interpret $G$ as 'is coronal’, then $f(G) = \{n\}$ and $(\forall x)G(x)$ is false, since $f(G) \subset D$. Likewise, $(\exists x)G(x)$ is true just in case the subset of $D$ that $f$ assigns to $G$ has at least one member. If we interpret $G$ as ‘is coronal’, then this is true, since $f$ maps $G$ to the subset $\{n\}$, which has one member, i.e. $|f(G)| \geq 1$.

\[\text{Recall that nasal sounds are produced with air flowing through the nose.}\]

\[\text{Coronal sounds are produced with the tip of the tongue.}\]
More complex expressions naturally get trickier. Combining quantified variables and constants is straightforward. For example, $(\forall x)H(a, x)$ is true just in case every member of $D$ can be the second member of each tuple in the set of ordered pairs that $f$ assigns to $H$, when $a$ is the first member. Thus, if $f(a) = m$, then $H(a, x) = \{\langle m, m \rangle, \langle m, n \rangle, \langle m, \eta \rangle\}$.

To get this set of ordered pairs, we might interpret $H(a, b)$ as saying that $a$ is at least as anterior as $b$, where anteriority refers to how far forward in the mouth the main constriction for the sound is. This basically means where the sound is made in the mouth. For example, the sound $[m]$ is made at the lips, $[t]$ is made with the tip of the tongue, and $[\eta]$ is made with the body of the tongue. On this model, $f(H) = \{\langle m, m \rangle, \langle m, n \rangle, \langle m, \eta \rangle, \langle n, n \rangle, \langle n, \eta \rangle, \langle \eta, \eta \rangle\}$.

We can use this latter interpretation of $H$ to treat another predicate logic formula: $(\forall x)H(x, x)$. Here there is still only one quantifier and no connectives, but there is more than one quantified variable. The same variable letter is used twice; the interpretation is that both arguments of $H$ must be the same. This expression is true if $H$ can pair all elements of $D$ with themselves. This is true in the just preceding case since $\{\langle m, m \rangle, \langle n, n \rangle, \langle \eta, \eta \rangle\} \subseteq f(H)$. That is, every sound in the set $D$ is at least as anterior as itself!

Let’s now consider how to interpret quantifiers and connectives in formulas together. The simplest case is where some connective occurs ‘outside’ any quantifier, e.g. $((\forall x)G(x) \land (\forall y)H(y))$. This is true just in case $f(G) = D$ and $f(H) = D$, that is, if $G$ is true of all the members of $D$ and $H$ is true of all the members of $D$, e.g. $(f(G) = D \land f(H) = D)$.

If the universal quantifier $\forall$ is ‘outside’ the connective, $(\forall x)(G(x) \land H(x))$, the formula ends up having the same truth value, but the interpretation is a little different. This latter formula is true on the model where $G$ and $H$ apply to every member of $D$. Here, we interpret the conjunction within the scope of the universal quantifier in its set-theoretic form: intersection. We then intersect $f(G)$ and $f(H)$. If this intersection is $D$, then the original expression is true. Thus $(\forall x)(G(x) \land H(x))$ is true just in case $(f(G) \cap f(H)) = D$ is true.\footnote{It is a theorem of set theory that if $A \cap B = U$, then $A = B = U$.}

Since expressions involving conjunction and universal quantification mean the same thing whether the quantifier is outside or inside the connective, it follows that a universal quantifier and be raised from or lowered into a conjunction changing the scope of the quantifier with no difference in truth
With the existential quantifier, scope forces different interpretations. The expression with the existential quantifier inside conjunction:

\((\exists x)G(x) \land (\exists y)H(y)\)

does not have the same value as the expression with the existential quantifier outside the conjunction:

\((\exists x)(G(x) \land H(x))\)

The first is true just in case there is some element of \(D\) that \(G\) holds of and there is some element of \(D\) that \(H\) holds of; the two elements need not be the same. In formal terms: \(|f(G)| \geq 1\) and \(|f(H)| \geq 1\), e.g. \((|f(G)| \geq 1 \land |f(H)| \geq 1)\). The two sets, \(f(G)\) and \(f(H)\), need not have any elements in common. The second is true only if there is some element of \(D\) that both \(G\) and \(H\) hold of, that is in \(f(G)\) and in \(f(H)\); that is, they must have at least one element in common: \(|(f(G) \cap f(H))| \geq 1\).

The point of these latter two examples is twofold. First, nesting connectives within the scope of quantifiers requires that we convert logical connectives to set-theoretic operations. Second, nesting quantifiers and connectives can result in different interpretations depending on the quantifier and depending on the connective.

The relative scope of more than one quantifier also matters. Consider the following four formulas.

\((5.1)\)

a. \((\forall x)(\forall y)G(x, y)\)

b. \((\exists x)(\exists y)G(x, y)\)

c. \((\forall x)(\exists y)G(x, y)\)

d. \((\exists x)(\forall y)G(x, y)\)

When the quantifiers are the same, their relative nesting is irrelevant. Thus (5.1a) is true just in case every element can be paired with every element, i.e. \(f(G) = D \times D\). Reversing the universal quantifiers gives exactly the same interpretation: \((\forall y)(\forall x)G(x, y)\). Likewise, (5.1b) is true just in case there’s at least one pair of elements in \(D\) that \(G\) holds of, i.e. \(|f(G)| \geq 1|\).
Reordering the existential quantifiers does not change this interpretation: 
\((\exists y)(\exists x)G(x, y)\).

On the other hand, when the quantifiers are different, the interpretation changes depending on which quantifier comes first. Example (5.1c) is true just in case every member of \(D\) can occur as the first member of at least one of the ordered pairs of \(f(G)\). Reversing the quantifiers produces a different interpretation. Thus \((\exists y)(\forall x)G(x, y)\) means that there is at least one element \(y\) that can occur as the second member of a pair with all elements.

These different interpretations can be depicted below. The required interpretation for (5.1c) with respect to our model is this:

\[(5.2) \ G(m, ?) \ G(n, ?) \ G(\eta, ?)\]

It doesn’t matter what the second member of each of these pairs is, as long as \(f(G)\) includes at least three with these first elements.

When we reverse the quantifiers, we must have one of the following three situations: a, b, or c.

\[(5.3) \ a. \ G(m, m) \ G(n, m) \ G(\eta, m) \]
\[b. \ G(m, n) \ G(n, n) \ G(\eta, n) \]
\[c. \ G(m, \eta) \ G(n, \eta) \ G(\eta, \eta) \]

All members of \(D\) must be paired with some unique member of \(D\).

The interpretation of (5.1d) is similar to the interpretation of (5.1c) with reversed quantifiers; it is true just in case there is some element that can be paired with every element of \(D\) as the second member of the ordered pair. We must have \(m, n, \) and \(\eta\) as the second member of the ordered pairs, but the first member must be the same across all three.

\[(5.4) \ a. \ G(m, m) \ G(m, n) \ G(m, \eta) \]
\[b. \ G(n, m) \ G(n, n) \ G(n, \eta) \]
\[c. \ G(\eta, m) \ G(\eta, n) \ G(\eta, \eta) \]

Some unique member of \(D\) must be paired with all members of \(D\).

Reversing the quantifiers in (5.1d), \((\forall y)(\exists x)G(x, y)\), is interpreted like this.
Every member of $D$ must occur as the second member of some pair in $f(G)$.

The interpretation of quantifier scope can get quite tricky in more complex WFFs.

Finally, notice that this system provides no interpretation for unquantified or free variables. Thus a syntactically well-formed expression like $G(x)$ has no interpretation. This would seem to be as it should be.

\section{5.3 Laws and Rules}

We can reason over formulas with quantifiers, but we need some additional Laws and Rules of Inference.

\subsection{5.3.1 Laws}

As we’ve already seen, quantifiers can distribute over logical connectives in various ways. The Laws of Quantifier Distribution capture the relationships between the quantifiers and conjunction and disjunction. We can separate these into logical equivalences and logical consequences. Here are the equivalences.

\begin{equation}
\text{(5.6) Laws of Quantifier Distribution: Equivalences}
\end{equation}

\begin{enumerate}
\item \textbf{Law 1: QDE1} \quad \neg(\forall x)\varphi(x) \iff (\exists x)\neg\varphi(x)
\item \textbf{Law 2: QDE2} \quad (\forall x)(\varphi(x) \land \psi(x)) \iff ((\forall x)\varphi(x) \land (\forall x)\psi(x))
\item \textbf{Law 3: QDE3} \quad (\exists x)(\varphi(x) \lor \psi(x)) \iff ((\exists x)\varphi(x) \lor (\exists x)\psi(x))
\end{enumerate}

Consider first Law 1 (QDE1). This says that that if $\varphi(x)$ is not universally true, then there must be at least one element for which $\varphi(x)$ is not true. This actually follows directly from what we have said above and the fact that the set-theoretic equivalent of negation is complement. The set-theoretic translation of $\neg(\forall x)\varphi(x)$ is $\neg(f(\varphi) = D)$. If this is true, then it follows that the complement of $f(\varphi)$ contains at least one element. The model-theoretic interpretation of the right side of the first law says just this: $f(\varphi)' \geq 1$. 

\begin{equation}
G(? , m) \ G(? , n) \ G(? , \eta)
\end{equation}
Law 2 (QDE2) allows us to move a universal quantifier down into a conjunction. The logic is that if something true for a whole set of predicates, then it is true for each individual predicate. Law 3 allows us to move an existential quantifier down into a disjunction. The logic is that if something is true for at least one of a set of predicates, then it is true for at least one of them, each considered independently. Both Law 1 and Law 2 should be expected given the example we went through in the previous section to explain the relationship of quantifiers and connectives. In general, the relationships above can be made sense of if we think of the universal quantifier as a conjunction of all the elements of the model $D$ and the existential quantifier as a disjunction of all the elements of the model $D$. Thus:

$$\forall x \in D, F(x) = \bigwedge_{i=1}^{\lvert D \rvert} F(x_i)$$

The big wedge symbol is interpreted as indicating that every element $x$ in the superscripted set $D$ is conjoined together. In other words, $(\forall x)F(x)$ is true just in case we apply $F$ to every member of $D$ and conjoin the values. The same is true for the existential quantifier:

$$\exists x \in D, F(x) = \bigvee_{i=1}^{\lvert D \rvert} F(x_i)$$

The big ‘v’ symbol is interpreted as indicating that every element $x$ in the superscripted set $D$ is disjoined together. In other words, $(\exists x)F(x)$ is true just in case we apply $F$ to every member of $D$ and disjoin the values.

Given the associativity of disjunction and conjunction (4.33), it follows that the universal quantifier can raise and lower into a conjunction and the existential quantifier can raise and lower into a disjunction.

For example, imagine that our universe $D$ is composed of only two elements $a$ and $b$. If would then follow that an expression like

$$\forall x (F(x) \land G(x))$$

is equivalent to
CHAPTER 5. PREDICATE LOGIC

\[(F(a) \land G(a)) \land (F(b) \land G(b))\]

Using Associativity, we can move terms around to produce

\[(F(a) \land F(b)) \land (G(a) \land G(b))\]

Translating each conjunct back, this is equivalent to

\[(∀x)F(x) \land (∀x)G(x))\]

Law 1 can also be cast in these terms, given DeMorgan’s Law (4.40). Thus

\[¬(∀x)F(x) ⇐⇒ (∃x)¬F(x)\]

is really the same thing as:

\[\left| D \right| \neg \bigwedge_{i=1}^{\left| D \right|} F(x_i) ⇐⇒ \bigvee_{i=1}^{\left| D \right|} ¬F(x_i)\]

The only difference is that the quantifiers range over a whole set of values from \(D\), not just a pair of values. Here are the logical consequences that relate to quantifier distribution.

(5.7) Laws of Quantifier Distribution: Consequences

Law 4: QDC4 \( (\forall x)\varphi(x) \lor (\forall x)\psi(x)) \implies (\forall x)(\varphi(x) \lor \psi(x))\)

Law 5: QDC5 \( (\exists x)(\varphi(x) \land \psi(x)) \implies ((\exists x)\varphi(x) \land (\exists x)\psi(x))\)

Law 4 (QDC4) says that a universal quantifier can be raised out of a disjunction. This is a logical consequence, not a logical equivalence. Thus, if we know that \((\forall x)G(x) \lor (\forall x)H(x))\), then we know \((\forall x)(G(x) \lor H(x))\), but not vice versa. For example, if we know that everybody in the room either all likes logic or all likes rock climbing, then we know that everybody in the room
either likes logic or likes rock climbing. However, if we know the latter, we cannot conclude the former. The latter is consistent with a situation where some people like logic, but other people like rock climbing. The former does not have this interpretation. Casting this in model-theoretic terms, if we have that \((f(G) = D \lor f(H) = D)\), then we have \((f(G) \cup f(H)) = D\).

Law 5 (QDC5) says that an existential quantifier can be lowered into a conjunction. As with Law 4, this is a logical consequence, not a logical equivalence. Thus, if we know \((\exists x)(G(x) \land H(x))\), then we know \((f(G) \land f(H))\), but not vice versa. For example, if we know that there is at least one person in the room that either likes logic or likes rock climbing, then we know that at least one person in the room likes logic or at least one person in the room likes rock climbing. The converse implication does not hold. From the fact that somebody likes logic or somebody likes rock climbing, it does not follow that that is the same somebody. Casting this in model-theoretic terms, if we have that \((f(G) \cap f(H)) \geq 1\), then we have \((f(G) \geq 1 \land f(H) \geq 1)\).

Now consider the Laws of Quantifier Scope. These govern when quantifier scope is and is not relevant. As above, there are logical equivalences and logical consequences. Here are the logical equivalences.

(5.8) Laws of Quantifier Scope: Equivalences

Law 6: QSE6 \((\forall x)(\forall y)\varphi(x, y) \iff (\forall y)(\forall x)\varphi(x, y)\)

Law 7: QSE7 \((\exists x)(\exists y)\varphi(x, y) \iff (\exists y)(\exists x)\varphi(x, y)\)

These are straightforward. The first (QSE6) says the relative scope of two universal quantifiers is irrelevant. The second (QSE7) says the relative scope of two existential quantifiers is irrelevant.

There is one logical consequence that relates to quantifier scope.

(5.9) Laws of Quantifier Scope: Consequences

Law 8: QSC8 \((\exists x)(\forall y)\varphi(x, y) \implies (\forall y)(\exists x)\varphi(x, y)\)

QSC8 reflects an implicational relationship between the antecedent and the consequent. The antecedent is true just in case there is some \(x\) that bears \(\varphi\) to every \(y\). The consequent is true just in case for every \(y\) there is at least one \(x\), not necessarily the same one, that it bears \(\varphi\) to.
5.3.2 Rules of Inference

There are four Rules of Inference for adding and removing quantifiers. You can use these to convert quantified expressions into simpler expressions with constants that our existing Laws and Rules will apply to, and then convert them back. These are thus extremely useful.

Universal Instantiation (U.I.) allows us to replace a universal quantifier with any arbitrary constant.

\[(\forall x)\varphi(x)\]
\[\therefore \varphi(c)\]

The intuition is that if \(\varphi\) is true of everything, then it is true of any individual thing we might cite.

Universal Generalization (U.G.) allows us to assume an arbitrary individual \(v\) and establish some fact about it. If something is true of \(v\), then it must be true of anything. Hence, \(v\) can be replaced with a universal quantifier. The intuition behind \(v\) is something like generic names like John/Jane Doe, Joe Sixpack, or Anyman. We use these designations to refer to a person who could be anybody.

\[\varphi(v)\]
\[\therefore (\forall x)\varphi(x)\]

The constant \(v\) is special; only it can be replaced with the universal quantifier. Other constants, \(a, b, c, \ldots\), cannot. The intuition here is that if we establish some property holds of an arbitrary individual, then it must hold of all individuals. Notice that Universal Generalization (5.11) can proceed only from \(v\), but Universal Instantiation (5.10) can instantiate to any constant, including \(v\).

Existential Generalization (E.G.) allows us to proceed from any constant to an existential quantifier.
CHAPTER 5. PREDICATE LOGIC

(5.12) Existential Generalization (E.G.)

\[
\varphi(c) \\
\therefore (\exists x)\varphi(x)
\]

Thus if some property holds of some specific individual, we can conclude that it holds of at least one individual.\(^6\)

Finally, Existential Instantiation (E.I.) allows us to go from an existential quantifier to a constant, as long as the constant has not been used yet in the proof. It must be a new constant.

(5.13) Existential Instantiation (E.I.)

\[
(\exists x)\varphi(x) \\
\therefore \varphi(w)
\]

where \(w\) is a new constant

This one is a bit tricky to state in intuitive terms. The basic idea is that if we know that some property holds of at least one individual, we can name that individual (as long as we don’t use a name we already know).

The intuition might be best understood with an example. Imagine we know that somebody among us has taken the last cookie and we’re trying to figure out who it is. We might start our inquiry like this: “We know that somebody has taken the cookie. For discussion, let’s call the culprit Bob…” That discussion could then proceed by discussing the clues we have, narrowing in on Bob’s identity. Notice that this kind of discussion could not proceed like this at all if one of the suspects was actually named Bob.

5.4 Proofs

Let’s now look at some simple proofs using this new machinery. First, we consider an example of Universal Instantiation. We prove \(H(a)\) from \(G(a)\) and \((\forall x)(G(x) \rightarrow H(x))\).

\(^6\)Here, we can proceed from any of the normal constants or from the arbitrary constant \(v\).
(5.14) 1 \( G(a) \) Given
2 \( (\forall x)(G(x) \rightarrow H(x)) \) Given
3 \( G(a) \rightarrow H(a) \) 2, U.I.
4 \( H(a) \) 1,3, M.P.

First, we remove the universal quantifier with Universal Instantiation and then use Modus Ponens to get the desired conclusion.

Next, we have an example of Universal Generalization. We try to prove \( (\forall x)(F(x) \rightarrow H(x)) \) from \( (\forall x)(F(x) \rightarrow G(x)) \) and \( (\forall x)(G(x) \rightarrow H(x)) \).

(5.15) 1 \( (\forall x)(F(x) \rightarrow G(x)) \) Given
2 \( (\forall x)(G(x) \rightarrow H(x)) \) Given
3 \( (F(v) \rightarrow G(v)) \) 1, U.I.
4 \( (G(v) \rightarrow H(v)) \) 2, U.I.
5 \( (F(v) \rightarrow H(v)) \) 3, 4 H.S.
6 \( (\forall x)(F(x) \rightarrow H(x)) \) 5 U.G.

First, we use Universal Instantiation on the two initial assumptions, massaging them into a form appropriate for Hypothetical Syllogism. We then use Universal Generalization to convert that result back into a universally quantified expression. Notice how we judiciously chose to instantiate to \( v \), anticipating that we would be using Universal Generalization later.

Finally, we consider a case of Existential Instantiation (E.I.). We prove \( ((\exists x)S(x) \land (\exists x)T(x)) \) from \( (\exists x)(S(x) \land T(x)) \).

(5.16) 1 \( (\exists x)(S(x) \land T(x)) \) Given
2 \( (S(a) \land T(a)) \) 1 E.I.
3 \( S(a) \) 2 Simp.
4 \( (\exists x)S(x) \) 3 E.G.
5 \( T(a) \) 2 Simp.
6 \( (\exists x)T(x) \) 5 E.G.
7 \( ((\exists x)S(x) \land (\exists x)T(x)) \) 4, 6 Conj.

First, we use Existential Instantiation to strip the quantifier and replace the variable with a new constant. We then split off the conjuncts with
Simplification and use Existential Generalization on each to add separate new existential quantifiers. We then conjoin the results with Conjunction.

Notice that the basic strategy in most of these proofs is fairly clear. Simplify the initial formulas so that quantifiers can be removed. Manipulate the instantiated formulas using the Laws and Rules from the preceding chapter. Finally, generalize to appropriate quantifiers.

The Laws allow us to prove things without first removing quantifiers. For example, in the following proof, we conclude \((\forall x)F(x) \land G(a)\) from \((\forall x)(F(x) \land G(x))\).

\[
\begin{align*}
(5.17) & \quad 1 \quad (\forall x)(F(x) \land G(x)) \quad \text{Given} \\
& \quad 2 \quad ((\forall x)F(x) \land (\forall x)G(x)) \quad 1 \text{ QDE2} \\
& \quad 3 \quad (\forall x)F(x) \quad 2 \text{ Simp.} \\
& \quad 4 \quad (\forall x)G(x) \quad 3 \text{ Simp.} \\
& \quad 5 \quad G(a) \quad 4 \text{ U.I.} \\
& \quad 6 \quad ((\forall x)F(x) \land G(a)) \quad 4,5 \text{ Conj.}
\end{align*}
\]

First, we use QDE2 to lower the universal quantifier into the conjunction. We extract each conjunct and then use Universal Instantiation on the second conjunct before re-conjoining.

Here’s a second example, where we prove \(\neg(\forall x)F(x)\) from \((\neg F(c) \land G(a))\).

\[
\begin{align*}
(5.18) & \quad 1 \quad (\neg F(c) \land G(a)) \quad \text{Given} \\
& \quad 2 \quad \neg F(c) \quad 1 \text{ Simp.} \\
& \quad 3 \quad (\exists x)\neg F(x) \quad 2 \text{ E.G.} \\
& \quad 4 \quad \neg(\forall x)F(x) \quad 3 \text{ QDE1}
\end{align*}
\]

Here we extract the first conjunct and use Existential Generalization to add an existential quantifier. We can then use QDE1 to move the negation outward and replace the existential with a universal quantifier.

### 5.4.1 Indirect Proof

We can use our other proof techniques with predicate logic too. Here we show how to use indirect proof. We prove \((\forall x)G(x) \rightarrow (\exists x)G(x))\) from no assumptions.
We start off by negating our conclusion and then attempting to produce a contradiction. The basic idea is to convert the negated conditional into a conjunction. We then extract the conjuncts. We use Universal Instantiation and then Existential Generalization on the first conjunct to produce a contradiction to the second conjunct. We conjoin the contradicting WFFs, completing the indirect proof.

5.4.2 Conditional Proof

We can prove the same thing by Conditional Proof. We assume the antecedent \((\forall x)G(x)\) and then attempt to prove the consequent \((\exists x)G(x)\) from it.

We begin by assuming the antecedent. From that assumption, we can instantiate to a constant and then generalize to an existential quantifier. That completes the proof.

5.5 Summary

In this chapter, we have treated the basics of predicate logic, covering syntax, semantics, and proof mechanisms.
We began with an introduction of the basic syntax of the system. Well-formed formulas of predicate logic (WFFs) are built on well-formed atomic statements. Atomic statements are built up from a finite alphabet of (lower-case) letters and a potentially infinite number of primes, e.g. \( p, q, r, p', q', q'' \), etc.

These, in turn, are combined via a restricted set of connectives into well-formed formulas. There are three important differences with respect to simple sentential logic. First, we have predicate symbols, e.g. \( F(a) \) or \( G(b, c) \), etc. In addition, we have the universal and existential quantifiers: \( \forall \) and \( \exists \). Finally, we have a notion of scope with respect to quantifiers which is important in the semantics of predicate logic.

The semantics of predicate logic is more complex than that of sentential logic. Specifically, formulas are true or false with respect to a model, where a model is a set of individuals and a mapping from elements of the syntax to sets of elements drawn from the set of elements in the model.

Quantifiers control how many individuals must be in the range of the mapping. For example, \( \exists x F(x) \) is true only if the predicate \( F \) is mapped to at least one individual; \( \forall x G(x) \) is true only if the predicate \( G \) is mapped to all individuals. These restrictions hold for any predicate in the scope of the quantifier with an as yet free variable.

All the Laws and Rules of Inference of sentential logic apply to predicate logic as well. However, there are additional Laws and Rules that govern quantifiers. The Laws of Quantifier Distribution and Scope govern the relations between quantifiers and between quantifiers and connectives. The Rules of Instantiation and Generalization govern how quantifiers can be added to or removed from formulas.

The chapter concluded with demonstrations of these various Laws and Rules in proofs. We also showed how Conditional Proof and Indirect Proof techniques are applicable in predicate logic.

### 5.6 Exercises

1. Identify the errors in the following WFFs:
   
   (a) \( (\forall x)G(y) \rightarrow H(x) \)
   
   (b) \( (\forall z)(F(z) \iff G(z)) \)
   
   (c) \( (F(x) \land G(y) \land H(z)) \)
2. In the following WFFs, mark the scope of each quantifier with labelled underlining.

(a) $(\forall x)(\forall y)(\forall z)F(a)$
(b) $((\forall x)F(y) \land (\exists y)G(x))$
(c) $(\exists x)((\forall y)F(x) \leftrightarrow F(y))$
(d) $(\exists z)(F(a) \land (F(b) \land (G(c) \rightarrow F(z))))$
(e) $\neg(\exists y)\neg\neg F(y)$

3. For the following questions, assume this model:

\[
D = \{\text{canto, cantas, canta, cantamos, cantan}\}
\]

\[
f(\text{FIRST}) = \{\text{canto, cantamos}\}
\]

\[
f(\text{SECOND}) = \{\text{cantas}\}
\]

\[
f(\text{THIRD}) = \{\text{canta, cantan}\}
\]

\[
f(\text{SG}) = \{\text{canto, cantas, canta}\}
\]

\[
f(\text{PL}) = \{\text{cantamos, cantan}\}
\]

For each of the following WFFs, indicate whether it is true or false with respect to this model.

(a) $(\exists x)\text{FIRST}(x)$
(b) $(\forall x)\text{FIRST}(x)$
(c) $(\forall x)(\text{SECOND}(x) \lor \text{PL}(x))$
(d) $(\forall y)(\text{SG}(x) \lor \text{PL}(x))$
(e) $(\exists z)(\text{SG}(z) \land \text{SECOND}(z))$

4. Prove $\neg(\forall z)(F(z) \land G(z))$ from $(\neg F(a) \lor \neg G(a))$.

5. Prove $G(a)$ from $(\exists x)(G(x) \land F(x))$. 
6. Prove \((\forall x)(\exists y)G(x, y)\) from \((\forall z)(\forall x)G(x, z)\).

7. Prove that \((\forall x)(G(x) \lor \neg G(x))\) is a tautology.

8. Prove that \(((\forall x)F(x) \rightarrow (\exists x)F(x))\) is a tautology.

9. Prove \(F(a)\) from \((\forall x)F(x)\) using Indirect Proof.

10. Prove \(\neg(\forall x)F(x)\) from \(\neg F(a)\) using Indirect Proof.

11. Use Conditional Proof to prove \(((\exists x)F(x) \rightarrow (\exists x)G(x))\) from the WFFs \(((\exists x)F(x) \rightarrow (\forall z)H(z))\) and \((H(a) \rightarrow G(b))\).

12. Construct a set of predicates and a model for a small area of language.
Chapter 6

Formal Language Theory

In this chapter, we introduce formal language theory, the computational theory of languages and grammars. Formal language theory is actually inspired by formal logic, enriched with insights from the theory of computation.

We begin with the definition of a language and then proceed to a rough characterization of the basic Chomsky hierarchy. We then turn to a more detailed consideration of the types of languages in the hierarchy and automata theory.

6.1 Languages

What is a language? Formally, a language $L$ is defined as as set (possibly infinite) of strings over some finite alphabet.

**Definition 7 (Language)** A language $L$ is a possibly infinite set of strings over a finite alphabet $\Sigma$.

We define $\Sigma^*$ as the set of all possible strings over some alphabet $\Sigma$. Thus $L \subseteq \Sigma^*$. The set of all possible languages over some alphabet $\Sigma$ is the set of all possible subsets of $\Sigma^*$, i.e. $2^{\Sigma^*}$ or $\wp(\Sigma^*)$. This may seem rather simple, but is actually perfectly adequate for our purposes.

Notice that $\Sigma^*$ is infinite, as there is no upper bound on the length of the strings that can be formed from $\Sigma$. 

96
6.2 Grammars

A grammar is a way to characterize a language $L$, a way to list out which strings of $\Sigma^*$ are in $L$ and which are not. If $L$ is finite, we could simply list the strings, but languages by definition need not be finite. In fact, all of the languages we are interested in are infinite. This is, as we showed in chapter 2, also true of human language.

Relating the material of this chapter to that of the preceding two, we can view a grammar as a logical system by which we can prove things. For example, we can view the strings of a language as WFFs. If we can prove some string $u$ with respect to some language $L$, then we would conclude that $u$ is in $L$, i.e. $u \in L$.

Another way to view a grammar as a logical system is as a set of formal statements we can use to prove that some particular string $u$ follows from some initial assumption. This, in fact, is precisely how we presented the syntax of sentential logic in chapter 4. For example, we can think of the symbol WFF as the initial assumption or symbol of any derivational tree of a well-formed formula of sentential logic. We then follow the rules for atomic statements (page 47) and WFFs (page 47).

Our notion of grammar will be more specific, of course. The grammar includes a set of rules from which we can derive strings. These rules are effectively statements of logical equivalence of the form: $\psi \rightarrow \omega$, where $\psi$ and $\omega$ are strings.\footnote{These statements seem to “go” in only one direction, yet they are not bound by the restriction we saw in first-order logic where a substitution based on logical consequence can only apply to an entire formula. It’s probably best to understand these statements as more like biconditionals, rather than conditionals, even though the traditional symbol here is the same as for a logical conditional.}

Consider again the WFFs of sentential logic. We know a formula like $(p \land q')$ is well-formed because we can progress upward from atomic statements to WFFs showing how each fits the rules cited above. For example, we know that $p$ is an atomic statement and $q$ is an atomic statement. We also know that if $q$ is an atomic statement, then so is $q'$. We also know that any atomic statement is a WFF. Finally, we know that two WFFs can be assembled together into a WFF with parentheses around the whole thing and a conjunction $\land$ in the middle.

We can also do this in the other direction, from WFF to atomic statements. We start with the assumption that we are dealing with a WFF. We
know first that a WFF can be composed of two separate WFFs surrounded by parentheses with a ∧ in the middle. We also know that those individual WFFs can be simple atomic statements and that one of those atomic statements can be p and the other q'.

The direction of the proof is thus irrelevant here.\(^2\)

We can represent all these steps in the form \(\psi \rightarrow \omega\) if we add some additional symbols. Let’s adopt \(W\) for a WFF and \(A\) for an atomic statement. If we know that \(p\) and \(q\) can be atomic statements, then this is equivalent to \(A \rightarrow p\) and \(A \rightarrow q\). Likewise, we know that any atomic statement followed by a prime is also an atomic statement: \(A \rightarrow A'\). We know that any atomic statement is a WFF: \(W \rightarrow A\). Last, we know that any two WFFs can be conjoined: \(W \rightarrow (W \land W)\).

\[
\begin{align*}
(6.1) \quad & A \rightarrow p \\
& A \rightarrow q \\
& A \rightarrow A' \\
& W \rightarrow A \\
& W \rightarrow (W \land W)
\end{align*}
\]

Each of these rules is part of the grammar of the syntax of WFFs. If every part of a formula follows one of the rules of the grammar of the syntax of WFFs, then we say that the formula is indeed a WFF.

Returning to the example \((p \land q')\), we can show that every part of the formula follows one of these rules by constructing a tree.

\[
\begin{align*}
(6.2) \quad & W \\
& \quad (W \land W) \\
& \quad \quad A \\
& \quad \quad A \\
& \quad \quad p \\
& \quad \quad A' \\
& \quad \quad q
\end{align*}
\]

\(^2\)It is quite relevant, however, if we are concerned with finding proofs in a simple fashion.
Each branch corresponds to one of the rules we posited. The mother of each branch corresponds to $\psi$ and the daughters to $\omega$. The elements at the very ends of branches—those without daughters—are referred to as terminal elements, and the elements higher in the tree are all non-terminal elements. If all branches correspond to actual rules of the grammar and the top node is a legal starting node, then the string given by the terminal elements is syntactically well-formed with respect to that grammar.

Formally, we define a grammar as $\{V_T, V_N, S, R\}$, where $V_T$ is the set of terminal elements, $V_N$ is the set of non-terminals, $S$ is a member of $V_N$, and $R$ is a finite set of rules of the form above. The symbol $S$ is defined as the only legal ‘root’ non-terminal: the designated ‘start’ symbol. As in the preceding example, we use capital letters for non-terminals and lowercase letters for terminals.

**Definition 8 (Grammar)** $\{V_T, V_N, S, R\}$, where $V_T$ is the set of terminal elements, $V_N$ is the set of non-terminals, $S$ is a member of $V_N$, and $R$ is a finite set of rules.

Looking more closely at $R$, we will require that the left side of a rule contain at least one non-terminal element and any number of other elements. We first define $\Sigma$ as $V_T \cup V_N$, all of the terminals and non-terminals together. $R$ is then a finite set of ordered pairs from $\Sigma^* V_N \Sigma^* \times \Sigma^*$. Thus $\psi \rightarrow \omega$ is equivalent to $\langle \psi, \omega \rangle$.

**Definition 9 (Rule)** $R$ is a finite set of ordered pairs from $\Sigma^* V_N \Sigma^* \times \Sigma^*$, where $\Sigma = V_T \cup V_N$.

We can now consider grammars of different types. The simplest case to consider first, from this perspective, are context-free grammars, or Type 2 grammars. In such a grammar, all rules of $R$ are of the form $A \rightarrow \psi$, where $A$ is a single non-terminal element of $V_N$ and $\psi$ is a string of terminals from $V_T$ and non-terminals from $V_N$.\(^3\) Such a rule says that a non-terminal $A$ can dominate the string $\psi$ in a tree. These are the traditional phrase-structure taught in introductory linguistics courses. The set of languages that can be generated with such a system is fairly restricted and derivations are straightforwardly represented with a syntactic tree. The partial grammar we exemplified above for sentential logic was of this sort.

\(^3\)Any ordering of terminals and non-terminals is valid on the right side of a context-free rule.
A somewhat more powerful system can be had if we allow context-sensitive rewrite rules, e.g. $A \rightarrow \psi/\alpha \beta$ (where $\psi$ cannot be $\epsilon$). Such a rule says that $A$ can dominate $\psi$ in a tree if $\psi$ is preceded by $\alpha$ and followed by $\beta$. If we set trees aside, and just concentrate on string equivalences, then this is equivalent to $\alpha A \beta \rightarrow \alpha \psi \beta$. Context-sensitive grammars are also referred to as Type 1 grammars.

In the other direction from context-free grammars, that is toward less powerful grammars, we have the regular or right-linear or Type 3 grammars. Such grammars only contain rules of the following form: $A \rightarrow xB$ or $A \rightarrow x$. The non-terminal $A$ can be rewritten as a single terminal element $x$ or a single non-terminal followed by a single terminal.

\begin{align*}
(6.3) & \quad 1 \text{ context-sensitive} \quad A \rightarrow \psi/\alpha \beta \\
& \quad 2 \text{ context-free} \quad A \rightarrow \psi \\
& \quad 3 \text{ right-linear} \quad \left\{ \begin{array}{l}
A \rightarrow xB \\
A \rightarrow x
\end{array} \right. \\
\end{align*}

We will see that these three types of grammars allow for successively more restrictive languages and can be paired with specific types of abstract models of computers. We will also see that the formal properties of the most restrictive grammar types are quite well understood and that as we move up the hierarchy, the systems become less and less well understood, or, more and more interesting.

Let’s look at a few examples. For all of these, assume the alphabet is $\Sigma = \{a, b, c\}$.

How might we define a grammar for the language that includes all strings composed of one instance of $b$ preceded by any number of instances of $a$: $\{b, ab, aab, aaab, \ldots\}$? We must first decide what sort of grammar to write among the three types we’ve discussed. In general, context-free grammars are the easiest and most intuitive to write. In this case, we might have something like this:

\begin{align*}
(6.4) & \quad S \rightarrow A b \\
& \quad A \rightarrow \epsilon \\
& \quad A \rightarrow A a
\end{align*}
The first rule takes the designated start symbol \( S \) and rewrites it as the non-terminal \( A \) followed by the terminal \( b \). The next two rules provide two different ways of rewriting \( A \). The rule \( A \rightarrow \epsilon \) says that \( A \) can be null, i.e. rewritten as \( \epsilon \). The last rule, \( A \rightarrow A \ a \) says that \( A \) can be rewritten as itself followed by an \( a \). In conjunction with the preceding rule, this allows for zero or more instances of \( a \).

This grammar is context-free because all rules have a single non-terminal on the left and a string of terminals and non-terminals on the right. This grammar cannot be right-linear because it includes rules where the right side has a non-terminal followed by a terminal, e.g. \( A \rightarrow A \ b \). This grammar cannot be context-sensitive because it contains rules where the right side is \( \epsilon \), e.g. \( A \rightarrow \epsilon \). For the strings \( b \), \( ab \), and \( aab \), this produces the following trees.

In terms of our formal characterization of grammars in Definition 8 above, we have:

\[
\begin{align*}
(6.5) & \quad V_T = \{a, b\} \\
& \quad V_N = \{S, A\} \\
& \quad S = S \\
& \quad R = \left\{ \begin{array}{l}
S \rightarrow A \ b \\
A \rightarrow \epsilon \\
A \rightarrow A \ a
\end{array} \right\}
\end{align*}
\]

Other grammars are possible for this language too. For example:
The first two rules rewrite $S$ as either $b$ or $A b$. The third and fourth rules provide options for expanding the non-terminal $A$ as either a single instance of $a$ or $A a$.

This grammar is context-free, but also qualifies as context-sensitive. We no longer have $\epsilon$ on the right side of any rule and a single non-terminal on the left qualifies as a string of terminals and non-terminals. This grammar produces the following trees for the same three strings.

We can also write a grammar that qualifies as right-linear that will characterize this language.

This produces trees as follows for our three examples.
There are, therefore, situations where the same language can be characterized in terms of all three types of grammars. This will not always be the case.

Let’s consider a somewhat harder case: a language where strings begin with an \( a \), end with a \( b \), with any number of intervening instances of \( c \), e.g. \{ab, acb, accb, \ldots\}. This can also be described using all three grammar types. First, a context-free grammar:

\[
\begin{align*}
S & \rightarrow a \ C \ b \\
C & \rightarrow C \ c \\
C & \rightarrow \epsilon
\end{align*}
\]

This grammar is neither right-linear nor context-sensitive. It produces trees like these:

Here is a right-linear grammar that generates the same strings:

\[
\begin{align*}
S & \rightarrow a \ C \\
C & \rightarrow c \ C \\
C & \rightarrow b
\end{align*}
\]

This produces trees as follows for the same three examples:
We can also write a grammar that is context-sensitive (and context-free) that produces this language.

\[
\begin{align*}
(6.15) \quad S & \rightarrow a \ b \\
S & \rightarrow a \ C \ b \\
C & \rightarrow C \ c \\
C & \rightarrow c
\end{align*}
\]

This results in the following trees.

We will see that the set of languages that can be described by the three types of grammar are not the same. Right-linear grammars can only accommodate a subset of the languages that can be treated with context-free and context-sensitive grammars. If we set aside the null string \( \epsilon \), context-free grammars can only handle a subset of the languages that context-sensitive grammars can treat.

In the following sections, we more closely examine the properties of the sets of languages each grammar formalism can accommodate and the set of abstract machines that correspond to each type.
6.3 Finite State Automata

In this section, we treat finite state automata. We consider two types of finite state automata: deterministic and non-deterministic. We define each formally and then show their equivalence.

What is a finite automaton? In intuitive terms, it is a very simple model of a computer. The machine reads an input tape which bears a string of symbols. The machine can be in any number of states and, as each symbol is read, the machine switches from state to state based on what symbol is read at each point. If the machine ends up in one of a set of particular states, then the string of symbols is said to be accepted. If it ends up in any other state, then the string is not accepted.

What is a finite automaton more formally? Let’s start with a deterministic finite automaton (DFA). A DFA is a machine composed of a finite set of states linked by arcs labeled with symbols from a finite alphabet. Each time a symbol is read, the machine changes state, the new state uniquely determined by the symbol read and the labeled arcs from the current state. For example, imagine we have an automaton with the structure in figure 6.17 below.

![Diagram of a DFA](6.17)

There are two states $q_0$ and $q_1$. The first state, $q_0$, is the designated start state and the second state, $q_1$, is a designated final state. This is indicated with a dark circle for the start state and a double circle for any final state. The alphabet $\Sigma$ is defined for this DFA as $\{a, b\}$.

This automaton describes the language where all strings contain an odd number of instances of the symbol $b$, for it is only with an input string that satisfies that restriction that the automaton will end up in state $q_1$. For example, let’s go through what happens when the machine reads the string $bab$. It starts in state $q_0$ and reads the first symbol $b$. It then follows the arc labeled $b$ to state $q_1$. It then reads the symbol $a$ and follows the arc from $q_1$ back to $q_1$. Finally, it reads the last symbol $b$ and follows the arc back to $q_0$. Since $q_0$ is not a designated final state, the string is not accepted.

Consider now a string $abb$. The machine starts in state $q_0$ and reads the symbol $a$. It then follows the arc back to $q_0$. It reads the first $b$ and follows
the arc to $q_1$. It reads the second $b$ and follows the arc labeled $b$ back to $q_0$. Finally, it reads the last $b$ and follows the arc from $q_0$ back to $q_1$. Since $q_1$ is a designated final state, the string is accepted.

We can define a DFA more formally as follows:

**Definition 10 (DFA)** A deterministic finite automaton (DFA) is a quintuple $\langle K, \Sigma, q_0, F, \delta \rangle$, where $K$ is a finite number of states, $\Sigma$ is a finite alphabet, $q_0 \in K$ is a single designated start state, and $\delta$ is a function from $K \times \Sigma$ to $K$.

For example, in the DFA in figure 6.17, $K = \{q_0, q_1\}$, $\Sigma = \{a, b\}$, $q_0$ is the designated start state and $F = \{q_1\}$. The function $\delta$ has the following domain and range:

<table>
<thead>
<tr>
<th>Domain</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0, a$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_0, b$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1, a$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1, b$</td>
<td>$q_0$</td>
</tr>
</tbody>
</table>

Thus, the function $\delta$ can be represented either graphically as arcs, as in (6.17), or textually as a table, as in (6.18).

4Some treatments distinguish deterministic automata from complete automata. A deterministic automaton has no more than one arc from any state labeled with any particular symbol. A complete automaton has at least one arc from every state for every symbol.

The situation of a finite automaton is a triple: $(x, q, y)$, where $x$ is the portion of the input string that the machine has already “consumed”, $q$ is the current state, and $y$ is the part of the string on the tape yet to be read. We can think of the progress of the tape as a sequence of situations licensed by $\delta$. Consider what happens when we feed $abab$ to the DFA in figure 6.17. We start with $(\epsilon, q_0, abab)$ and then go to $(a, q_0, bab)$, then to $(ab, q_1, ab)$, etc. The entire derivation are encoded with the turnstile symbol $\vdash$. The derivation is given below:

(6.19) $(\epsilon, q_0, abab) \vdash (a, q_0, bab) \vdash (ab, q_1, ab) \vdash (aba, q_1, b) \vdash (abab, q_0, \epsilon)$
Since the DFA does not end up in a state of \( F \) \( (q_0 \not\in F) \), this string is not accepted.

Let’s define the turnstile more formally as follows:

**Definition 11 (produces in one move)** Assume a DFA \( M = \langle K, \Sigma, \delta, q_0, F \rangle \). A situation \((x, q, y)\) produces situation \((x', q', y')\) in one move iff: 1) there is a symbol \( \sigma \in \Sigma \) such that \( y = \sigma y' \) and \( x' = x \sigma \) \( (i.e., \ the \ machine \ reads \ one \ symbol) \), and 2) \( \delta(q, \sigma) = q' \) \( (i.e., \ the \ appropriate \ state \ change \ occurs \ on \ reading \ \sigma) \).

The basic idea is that \( x \) is converted to \( x' \) by adding \( \sigma \), \( y \) is converted to \( y' \) by removing \( \sigma \), and there is an arc from \( q \) to \( q' \) labeled with \( \sigma \).

We can use this to define a more general notion “produces in zero or more steps”: \( \vdash^* \). We say that \( S_1 \vdash^* S_n \) if there is a sequence of situations \( S_1 \vdash S_2 \vdash \ldots \vdash S_{n-1} \vdash S_n \). Thus the derivation in (6.19) is equivalent to the following.

\[
(6.20) \quad (\epsilon, q_0, abab) \vdash^* (abab, q_0, \epsilon)
\]

Let’s now consider non-deterministic finite automata (NFAs). These are just like DFAs except i) arcs can be labeled with the null string \( \epsilon \), and ii) there can be multiple arcs with the same label from the same state; thus \( \delta \) is a relation, not a function, in a NFA.

**Definition 12 (NFA)** A non-deterministic finite automaton \( M \) is a quintuple \( \langle K, \Sigma, \Delta, q_0, F \rangle \), where \( K \), \( \Sigma \), \( q_0 \), and \( F \) are as for a DFA, and \( \Delta \), the transition relation, is a finite subset of \( K \times (\Sigma \cup \epsilon) \times K \).

Let’s look at an example. The NFA in (6.21) generates the language where any instance of the symbol \( a \) must have at least one \( b \) on either side of it; the string must also begin with at least one instance of \( b \).
Here, \( q_0 \) is the designated start state and \( q_1 \) is in \( F \). We can see that there are two arcs from \( q_0 \) on \( b \), but none on \( a \); this automaton is thus necessarily non-deterministic.

The transition relation \( \Delta \) can be represented in tabular form as well. Here, we list all the mappings for every combination of \( K \times \Sigma \).

\[
\begin{array}{c|c}
\text{domain} & \text{range} \\
q_0, a & \emptyset \\
q_0, b & \{q_0, q_1\} \\
q_1, a & \{q_0\} \\
q_1, b & \{q_0\}
\end{array}
\]

Notice how \( K \times \Sigma \) maps to a (possibly empty) set of states.

Given that there are multiple paths through an NFA for any particular string, how do we assess whether a string is accepted by the automaton? To see if some string is accepted by a NFA, we must determine if there is at least one path through the automaton that terminates in a state of \( F \).

Consider the automaton above and the string \( bab \). There are several paths to consider.

\[
\begin{align*}
(6.23) & \quad \text{a. } (\epsilon, q_0, bab) \vdash (b, q_0, ab) \vdash ? \\
 & \quad \text{b. } (\epsilon, q_0, bab) \vdash (b, q_1, ab) \vdash (ba, q_0, b) \vdash (bab, q_0, \epsilon) \\
 & \quad \text{c. } (\epsilon, q_0, bab) \vdash (b, q_1, ab) \vdash (ba, q_0, b) \vdash (bab, q_1, \epsilon)
\end{align*}
\]

The first, (6.23a), doesn’t terminate. The second terminates, but only in a non-final state. The third, (6.23c), terminates in a final state. Hence, since there is at least one path that terminates in a final state, the string is accepted.

It’s a little trickier when the NFA contains arcs labeled with \( \epsilon \). For example:

\[
\begin{array}{c}
\begin{array}{c}
\text{(6.24)} \\
q_0 \\
\text{a} \\
\epsilon \\
\text{b} \\
q_1 \\
\text{b} \\
q_2
\end{array}
\end{array}
\]
Here we have the usual sort of non-determinism with two arcs labeled with $a$ from $q_0$. We also have an arc labeled $\epsilon$ from $q_1$ to $q_0$. This latter sort of arc can be followed at any time without consuming a symbol. Let’s consider how a string like $aba$ might be parsed by this machine. The following chart shows all possible paths.

\[(6.25)\]

\begin{align*}
\text{a. } (\epsilon, q_0, aba) &\vdash (a, q_0, ba) \vdash (ab, q_0, a) \vdash (aba, q_1, \epsilon) \\
\text{b. } (\epsilon, q_0, aba) &\vdash (a, q_1, ba) \vdash (ab, q_1, a) \vdash (aba, q_1, \epsilon) \\
\text{c. } (\epsilon, q_0, aba) &\vdash (a, q_1, ba) \vdash (ab, q_0, a) \vdash (aba, q_0, \epsilon) \\
\text{d. } (\epsilon, q_0, aba) &\vdash (a, q_1, ba) \vdash (ab, q_1, a) \vdash (aba, q_0, \epsilon) \\
\text{e. } (\epsilon, q_0, aba) &\vdash (a, q_0, ba) \vdash (ab, q_0, a) \vdash (aba, q_0, \epsilon) \\
\text{f. } (\epsilon, q_0, aba) &\vdash (a, q_1, ba) \vdash (ab, q_1, a) \vdash (aba, q_1, \epsilon) \\
\text{g. } (\epsilon, q_0, aba) &\vdash (a, q_0, ba) \vdash (ab, q_0, a) \vdash (aba, q_0, \epsilon) \\
\text{h. } (\epsilon, q_0, aba) &\vdash (a, q_1, ba) \vdash (a, q_0, ba) \vdash (ab, q_0, a) \vdash (aba, q_1, \epsilon) \\
\text{i. } (\epsilon, q_0, aba) &\vdash (a, q_1, ba) \vdash (ab, q_1, a) \vdash (aba, q_0, \epsilon) \\
\text{j. } (\epsilon, q_0, aba) &\vdash (a, q_1, ba) \vdash (ab, q_1, a) \vdash (aba, q_0, \epsilon)
\end{align*}

The $\epsilon$-arc can be followed whenever the machine is in state $q_1$. It is indicated in the chart above by a move from $q_1$ to $q_0$ without a symbol being read. Note that it results in an explosion in the number of possible paths. In this case, since at least one path ends up in the designated final state $q_1$, the string is accepted.

Notice that it’s a potentially very scary proposition to determine if some NFA generates some string $x$. Given that there are $\epsilon$-arcs, which can be followed at any time without reading a symbol, there can be an infinite number of paths for any finite string.\footnote{This can arise if we have cycles involving $\epsilon$.} Fortunately, this is not a problem, because DFAs and NFAs generate the same class of languages.

**Theorem 1** DFAs and NFAs produce the same languages.

Let’s show this. DFAs are obviously a subcase of NFAs; hence any language generated by a DFA is trivially generated by an NFA.
Proving this in the other direction is a little trickier. What we will do is show how a DFA can be constructed from any NFA (Hopcroft and Ullman, 1979). Recall that the arcs of an NFA can be represented as a map from \( K \times (\Sigma \cup \epsilon) \) to all possible subsets of \( K \). What we do to construct the DFA is to use these sets of states as literal labels for new states.

For example, in the NFA in (6.21), call it \( M \), we have \( \Delta \) as in (6.22). The possible sets of states are: \( \emptyset, \{q_0\}, \{q_1\}, \text{ and } \{q_0, q_1\} \). The new DFA \( M' \) will then have state labels: \( \emptyset, [q_0], [q_1], \text{ and } [q_0, q_1] \), replacing the curly braces that denote sets with square braces which we will use to denote state labels. For the new DFA, we define \( \delta' \) as follows:

\[
\delta'([q_1, q_2, \ldots, q_n], a) = [p_1, p_2, \ldots, p_n]
\]

if and only if, in the original NFA:

\[
\Delta([q_1, q_2, \ldots, q_n], a) = \{p_1, p_2, \ldots, p_n\}
\]

The latter means that we apply \( \Delta \) to every state in the first list of states and union together the resulting states.

Applying this to the NFA in (6.21), we get this chart for the new DFA.

\[
\begin{align*}
\delta([\emptyset], a) & = [\emptyset] \\
\delta([\emptyset], b) & = [\emptyset] \\
\delta([q_0], a) & = [\emptyset] \\
\delta([q_0], b) & = [q_0, q_1] \\
\delta([q_1], a) & = [q_0] \\
\delta([q_1], b) & = [q_0] \\
\delta([q_0, q_1], a) & = [q_0] \\
\delta([q_0, q_1], b) & = [q_0, q_1]
\end{align*}
\]

The initial start state was \( q_0 \), so the new start state is \( [q_0] \). Any set containing a possible final state from the initial automaton is a final state in the new automaton: \( [q_1] \) and \( [q_0, q_1] \). The new automaton is given below.

\[\text{Recall that there will be } 2^K \text{ of these if there are } K \text{ states.}\]
This automaton accepts exactly the same language as the previous one. If we can always construct a DFA from an NFA that accepts exactly the same language, it follows that there is no language accepted by an NFA that cannot be accepted by a DFA. \(\square\)

Notice two things about the resulting DFA in (6.27). First, there is a state that cannot be reached: \([q_1]\). Such states can safely be pruned. The following automaton is equivalent to (6.27).

Second, notice that the derived DFA can, in principle, be massively bigger than the original NFA. In the worst case, if the original NFA has \(n\) states, the new automaton can have as many as \(2^n\) states.\(^7\)

In the following, since NFAs and DFAs are equivalent, I will refer to the general class as *Finite State Automata* (FSAs).

\(^7\)There are algorithms for minimizing the number of states in a DFA, but they are beyond the scope of this introduction. See Hopcroft and Ullman (1979). Even minimized, it is generally true that an NFA will be smaller than its equivalent DFA.
6.4 Regular Languages

We now consider the class of regular languages. We’ll show that these are precisely those that can be accepted by an FSA and which can be generated by a right-linear grammar.

The regular languages are defined as follows.

**Definition 13 (Regular Language)** Given a finite alphabet \( \Sigma \):

1. \( \emptyset \) is a regular language.
2. For any string \( x \in \Sigma^* \), \( \{x\} \) is a regular language.
3. If \( A \) and \( B \) are regular languages, then so is \( A \cup B \).
4. If \( A \) and \( B \) are regular languages, then so is \( AB \).
5. If \( A \) is a regular language, then so is \( A^* \).
6. Nothing else is a regular language.

Consider each of these operations in turn. First, we have that any string of symbols from the alphabet can be a specification of a language. Thus, if the alphabet is \( \Sigma = \{a, b, c\} \), then the regular language \( L \) can be \( \{a\} \).

If \( L_1 = \{a\} \) and \( L_2 = \{b\} \), then we can define the regular language which is the union of \( L_1 \) and \( L_2 \): \( L_3 = L_1 \cup L_2 \), i.e. \( L_3 = \{a, b\} \). In string terms, this is usually written \( L_3 = (a|b) \).

We can also concatenate two regular languages, e.g. \( L_3 = L_1L_2 \), e.g. \( L_3 = \{ab\} \).

Finally, we have Kleene star, which allows us to repeat some regular language zero or more times. Thus, if \( L_1 \) is a regular language, then \( L_2 = L_1^* \) is a regular language, e.g. \( L_2 = \{a, aa, aaa, \ldots\} \). In string terms: \( L_2 = a^* \).

These operations can, of course, be applied recursively in any order. For example, \( a(b^*|c)a^* \) refers to the language where all strings are composed of a single instance of \( a \) followed by any number of instances of \( b \) or a single \( c \), followed in turn by any number of instances of \( a \). We can construct this language stepwise from the definition above.

---

8 Notice that it would work just as well to start form any symbol \( a \in \Sigma \) here, since we have recursive concatenation in #4.
9 In complex examples, we can use parentheses to indicate scope.
(6.29) \[ a \] clause #2
\[ b \] clause #2
\[ c \] clause #2
\[ b^* \] clause #5
\[ (b^*|c) \] clause #3
\[ a(b^*|c) \] clause #4
\[ a^* \] clause #5
\[ a(b^*|c)a^* \] clause #4

We can go in the other direction as well. For example, how might we describe the language where all strings contain an even number of instances of \( a \) plus any number of the other symbols: \( ((b|c)^*a(b|c)^*a(b|c)^*)^* \).

6.5 Automata and Regular Languages

It turns out that the set of languages that can be accepted by a finite state automaton is exactly the regular languages.

**Theorem 2** A set of strings is a finite automaton language if and only if it is a regular language.

We won’t prove this rigorously, but we can see the logic of the proof fairly straightforwardly. There are really only four things that we can do with a finite automaton, and each of these four corresponds to one of the basic clauses of the definition of a regular language.

First, we have that a single symbol is a legal regular language because we can have a finite automaton with a start state, a single arc, and a final state.

Second, we have concatenation of two regular languages by taking two automata and connecting them with an arc labeled with \( \epsilon \).
We connect all final states of the first automaton with the start state of the second automaton with $\epsilon$-arcs. The final states of the first automaton are made non-final. The start state of the second automaton is made a non-start state.

Union is straightforward as well. We simply create a new start state and then create arcs from that state to the former start states of the two automata labeled with $\epsilon$. We create a new final state as well, with $\epsilon$-arcs from the former final states of the two automata.

Finally, we can get Kleene star by creating a new start state (which is also a final state), a new final state, and an $\epsilon$-loop between them.

If we can construct an automaton for every step in the construction of a regular language, it should follow that any regular language can be accepted by some automaton.\textsuperscript{10}

\textsuperscript{10}A rigorous proof would require that we go through this in the other direction as well, from automaton to regular language.
6.6 Right-linear Grammars and Automata

Another equivalence that is of use is that between the regular languages and right-linear grammars. Right-linear grammars generate precisely the set of regular languages.

We can show this by pairing the rules of a right-linear grammar with the arcs of an automaton. First, for every rewrite rule of the form $A \rightarrow x B$, we have an arc from state $A$ to state $B$ labeled $x$. For every rewrite rule of the form $A \rightarrow x$, we have an arc from state $A$ to the designated final state, call it $F$.

Consider this very simple example of a right-linear grammar.

\[(6.34)\]

a. $S \rightarrow a A$

b. $A \rightarrow a A$

c. $A \rightarrow a B$

d. $B \rightarrow b$

This generates the language where all strings are composed of two or more instances of $a$, followed by exactly one $b$.

If we follow the construction of the FSA above, we get this:

\[(6.35)\]

This FSA accepts the same language generated by the right-linear grammar in (6.34).

Notice now that if FSAs and right-linear grammars generate the same set of languages and FSAs generate regular languages, then it follows that right-linear grammars generate regular languages. Thus we have a three-way equivalence between regular languages, right-linear grammars, and FSAs.

6.7 Closure Properties of Regular Languages

Let’s now turn to closure properties of the regular languages. A closure property is one that when applied to an element of a set produces an element
of the same set. In the present context, a closure property for the regular languages is one that, when applied to a regular language, produces a regular language.

By the definition of regular language, it follows that they are closed under the properties that define them: concatenation, union, Kleene star. They are also closed under complement. The complement of some regular language $L$ defined over the alphabet $\Sigma$ is $L' = \Sigma^* - L$, i.e. all strings over the same alphabet not in $L$.

It’s rather easy to show this using DFAs. In particular, to construct the complement of some language $L$, we create the DFA that generates that language and then swap the final and non-final states.

Let’s consider the DFA in (6.17) on page 105 above. This generates the language $a^*ba^*(ba^*ba^*)^*$, where every legal string contains an odd number of instances of the symbol $b$, and any number of instances of the symbol $a$. We now reverse the final and non-final states so that $q_0$ is both the start state and the final state.

This now generates the complement language: $a^*(ba^*ba^*)^*$. Every legal string has an even number of instances of $b$ (including zero), and any number of instances of $a$.

With complement so defined, and DeMorgan’s Law (the set-theoretic version), it follows that the regular languages are closed under intersection as well. Recall the following equivalences from chapter 3.

\[(X \cup Y)' = X' \cap Y'\]
\[(X \cap Y)' = X' \cup Y'\]

Therefore, since the regular languages are closed under union and under complement, it follows that they are closed under intersection. Thus if we want to intersect the languages $L_1$ and $L_2$, we union their complements, i.e. $L_1 \cap L_2 = (L_1' \cup L_2')'$. 

\[\text{(6.36)}\]
6.8 Pumping Lemma for Regular Languages

The Pumping Lemma for regular languages is quite daunting in its full formal glory, but it is a powerful tool for classifying languages and actually quite intuitive. In this section, we build up to, explain, and exemplify it.

Recall that the regular languages are defined by three operations: concatenation, union, and Kleene star. The last is actually rather different from the first two: it is only with Kleene star that a regular language can be infinite.

Concatenation cannot produce an infinitely large language from finite languages. For example, concatenating the language \( L_1 = \{a, b\} \) with the language \( L_2 = \{b, c\} \) produces a larger but still finite language where \( L_3 = \{ab, ac, bb, bc\} \). Here the size of the concatenated language is equal to the product of the sizes of the concatenated languages: \( |L_3| = |L_1| \times |L_2| \). Thus, if \( L_1 \) and \( L_2 \) are finite, \( L_3 \) must be finite.

Likewise, union can only produce a finite language from finite languages. For example, the union of \( L_1 = \{a\} \) and \( L_2 = \{b\} \) gives: \( L_3 = \{a, b\} \), a larger language, but still finite. Here, the size of the unioned language is no bigger than the sum of the sizes of the individual languages: \( |L_3| \leq |L_1| + |L_2| \). Thus, if \( L_1 \) and \( L_2 \) are finite, \( L_3 \) must be finite.

Kleene star, on the other hand, produces an infinitely large language from any language, whether it is already infinite or not. For example, if \( L_1 = \{a\} \), a finite language, then applying Kleene star to it produces the infinite language \( L_2 = \{a, aa, aaa, \ldots\} \).

Therefore it follows that if a language is infinite and regular, then its characterization must include Kleene star. Since Kleene star allows a substring (or sublanguage) to recur an infinite number of times, it follows that in an infinite regular language there will be at least one substring that can be repeated an infinite number of times. This is what the Pumping Lemma says:

**Theorem 3** If \( L \) is a regular language, then there is a constant \( n \) such that if \( z \) is any word in \( L \), and \( |L| > n \), then we can write \( z = uvw \) such that \( |uv| \leq n \), \( |v| \geq 0 \), and for all \( i \geq 0 \), \( uv^i w \) is in \( L \). In addition, \( n \) is no greater than the number of states in the smallest FSA that accepts \( L \).\(^{11}\)

\(^{11}\)The language of the text has it that this is a lemma, not a theorem, but the distinction is not important. We therefore give it as a theorem, but continue the traditional characterization as a lemma.
This says that if a language is regular and we look at words whose length exceeds a specific length, then those words will always have a substring that can be repeated any number of times still producing a word of the language.

Consider for example, the language $L = \{ac, abc, abbc, abbbbc, \ldots\}$, or $ab^*c$. Here, once we consider strings of at least three letters, there is a repeatable substring. Thus, if $n = 3$ and $z = abc$, then $u = a$, $v = b$, and $w = c$. Any number of repetitions of $b$ results in a legal word.

Notice that if $n$ were set to 2, we would not always find a repeating substring. Thus, in $ac$ there is no $v$ can that be repeated.

Notice too that the repetitions do not necessarily enumerate the entire language. For example, $n = 3$ for the language $a(b|c)^*d$. Thus, in the string $abd$, $u = a$, $v = b$, and $w = d$. The substring $b$ can be repeated any number of times. However, the language also allows any number of repetitions of $c$ in the same position.

Since Kleene star can be used more than once in the definition of a language, there may be more than one point in a string that can qualify for $v$. Consider the language $ab^*c^*d$. Here, $n = 3$ as well, thus in a string $abd$, $b$ can be repeated and in a string $acd$, $c$ can be repeated. However, in a string $abcd$, either $b$ or $c$ can be repeated.

Finally, the repeated string can of course be longer than a single letter. In the language $a(bc)^*d$, $n = 4$. In a string $abcd$, $u = a$, $v = bc$, and $w = d$.

What’s important about the Pumping Lemma is that it can be used to show that some language is not regular. Consider for example, the language $a^n$, where $n$ is prime, i.e. $\{a, aa, aaa, aaaaa, aaaaaaa, aaaaaaaaaa, \ldots\}$. For any $n$ we choose, there is no way to parse the string as $uvw$.

Imagine $n = 1$ and we start with $a$. Then $u = \epsilon$, $v = a$, and $w = \epsilon$. This won’t work because $uv^3w = aaaa$, which is not in $L$.

Imagine $n = 2$ and we start with $aa$. Then $u = a$, $v = a$, and $w = \epsilon$. This won’t work either because $uv^3w = aaaa$, which is not in $L$.

In fact, with sufficient mathematical expertise, we can show that there is no finite $n$ that will work. Thus $a^n$, where $n$ is prime, is not regular.

Here’s another example: the language $a^nb^n$, i.e. $\{\epsilon, ab, aabb, aaabbb, \ldots\}$. Once again, it is impossible to find a workable value for $n$.

Imagine $n = 1$ and we start with $ab$. The key question is what qualifies as $v$? There are only three possibilities:
Each of these parses leads to incorrect predictions. On the first choice, we cannot set $i$ to 2, as $uv^2w = aab$ is not in the language. On the second choice, we also cannot set $i$ to 2, as $uv^2w = abb$ is not in the language. Finally, the third choice also does not allow $i$ as 2, since $uv^2w = abab$ is not in the language.

The problem persists for larger values of $n$; there is no way to find a finite value of $n$ that works, given the way the language is defined. Hence $a^n b^n$ is not regular.

Finally, notice from the definition that the size of $n$ is a function of the size of the automaton that accepts the language. Specifically, $n$ is no bigger than the smallest FSA that will accept the language. We leave the explanation of this as an exercise.

### 6.9 Relevance of Regular Languages

Do the regular languages and FSAs have any interest for linguists? Yes. It’s been claimed that phonology can be treated as a finite-state system. It’s also been claimed that morphology is finite state. We briefly discuss each of these in this section.

It’s been argued that phonological generalizations can be characterized in terms of FSAs (Karttunen, 1983; Koskenniemi, 1984). There are actually two main approaches: static phonological generalizations (Bird and Ellison, 1994; Bird, 1995) and phonological relationships (Kaplan and Kay, 1994). We discuss the former here and return to the latter in the next chapter.

Consider a very simple generalization with respect to Syllable Structure, e.g. all syllables in Hawaiian are composed of a single consonant followed by

---

12Syllable structure can be expressed as a context-free grammar. Is syllable structure necessarily context-free?
13If morphology is finite state, then attempts to treat morphological structure using the full power of syntactic theory would appear to be bringing too much computational power to the task.
a single vowel. This is very easy to state as a regular expression. Let us say, for simplicity, that the consonants of Hawaiian are \{p, t, k, r, n\} and the vowels are \{a, i, u\}.\(^{14}\) This means that Hawaiian has possible words of the form: CVCVCV. . . . We can express this restriction as a regular expression.

\[(6.39) \ (p|t|k|r|n)(a|i|u)((p|t|k|r|n)(a|i|u))^*\]

All words must be composed of a sequence of any consonant followed by any vowel followed by any number of repetitions of the same sequence. Notice how the regular expression includes the stipulation that all words are at least one syllable long.

Another fairly common kind of phonological generalization is Vowel Harmony, e.g. in Hungarian, Finnish, or Turkish. In vowel harmony systems all the vowels of a language are partitioned into some number of groups and the words of the language are restricted so that all the vowels of any one word must be drawn exclusively from only one group. For example, imagine the vowels of Turkish are \{i, a, e, u, ü, ö\}.\(^{15}\) Let’s assume that the consonants are the same as in Hawaiian and that there are two groups of vowels: \{i, a, e\} and \{u, ü, ö\}. We can then express the vowel harmony restriction as follows. Some words are of this form:

\[(6.40) \ \text{“Type 1”} = (p|t|k|r|n)^*(i|a|e)(p|t|k|r|n)^*(i|a|e)(p|t|k|r|n)^*\]

Other words are of this form:

\[(6.41) \ \text{“Type 2”} = (p|t|k|r|n)^*(u|ü|ö)(p|t|k|r|n)^*(u|ü|ö)(p|t|k|r|n)^*\]

All words then are of this sort:

\[(6.42) \ (“Type 1” | “Type 2”)\]

\(^{14}\)Hawaiian actually has more consonants and vowels than this, but we’re simplifying the inventory to make the general point.

\(^{15}\)Again, Turkish has many more vowels than this and the vowel harmony system of Turkish is more complex. The vowels marked with diaeresis (two dots) are vowels produced in the front of the mouth with lip rounding. The vowel [ü] corresponds to [i] with rounding and the vowel [ö] corresponds to [e] with rounding.
The basic idea is that each word type is defined as being composed of instances of vowels from the relevant group (at least one vowel) with any number of interspersed consonants. The words of the language follow one or the other pattern.

Morphological generalizations have also been proposed to be instances of regular grammars. For example, the set of possible verbal inflectional affixes in English might be cast as a regular expression:

\[(6.43) \text{VERB(-s|ed|-ing|)}\]

The system rapidly gets more complex, however. For example, recall the pattern of forming words in English with \(-ize\), \(-ation\), and \(-al\) discussed on page 25 in Chapter 2. This is not so obviously regular.

\section*{6.10 Summary}

This chapter began with a formal definition of a language as a set of strings over a finite alphabet (or vocabulary). There are finite and infinite languages and we focus on infinite languages.

Such a language can be described with a grammar, where a grammar is defined as a finite set of symbols and a finite set of rules mapping strings of symbols to other strings of symbols. We considered three general categories of grammar: context-sensitive, context-free, and right-linear. These form a hierarchy and are successively more restrictive, with right-linear being the most restrictive.

We next turned to automata, abstract models of computers. A finite state automaton has a finite number of states and a finite number of arcs between states labeled with the symbols of an alphabet. FSAs come in two flavors: deterministic and non-deterministic. We showed how these are equivalent in terms of the kinds of languages they can describe.

A finite state automaton can describe a particular class of languages: the regular languages. These are the set of languages that can be formalized in terms of only union, concatenation, or Kleene star. We also showed how regular languages can be described in terms of right-linear grammars.

Lastly, we considered other closure properties of the regular languages: complement and intersection and went over how the Pumping Lemma could be used to prove that some language is not regular.
The chapter closed with a discussion of the relevance of regular languages to morphology and phonology.

6.11 Exercises

1. Write a right-linear grammar that generates the language where strings must have exactly one instance of $a$ and any number of instances of the other symbols to either side ($\Sigma = \{a, b, c\}$).

2. Write a right-linear grammar where strings must contain an $a$ and a $b$ in just that order, plus any number of other symbols ($\Sigma = \{a, b, c\}$) before, after, and in between.

3. Write a context-free grammar where legal strings are composed of some number of instances of $a$, followed by a $c$, followed by exactly the same number of instances of $b$ as there were of $a$, followed by another $c$.

4. Write a context-sensitive grammar where legal strings are composed of some number of instances of $a$, followed by exactly the same number of instances of $b$ as there were of $a$, followed by exactly the same number of instances of $c$.

5. Write a DFA that generates the language where strings must have exactly one instance of $a$ and any number of instances of the other symbols ($\Sigma = \{a, b, c\}$).

6. Write a DFA where strings must contain an $a$, a $b$, and a $c$ in just that order, plus any number of other symbols ($\Sigma = \{a, b, c\}$).

7. Write a DFA where anywhere $a$ occurs, it must be immediately followed by a $b$, and any number of instances of $c$ may occur around those bits.

8. Describe this language in words: $b(a^*c^*)c$

9. Describe this language in words: $b(a|c)^*c$

10. Describe this language in words: $(a|b|c)^*a(a|b|c)^*a(a|b|c)^*$

11. Formalize this language as a regular language: all strings contain precisely three symbols ($\Sigma = \{a, b, c\}$).
12. Explain why $ww^R$ cannot be regular.

13. Is the following a regular language? All strings contain more instances of $a$ than of $b$, in any order, with no instances of $c$. If it is, give a regular expression or FSA; if it is not, explain why.

14. Why is the size of a unioned language: $|L_3| \leq |L_1| + |L_2|$? Under what circumstances does $|L_3| = |L_1| + |L_2|$?

15. The size of $n$ in the Pumping Lemma is given as less than the number of states in the smallest FSA that can describe the language. Explain why this restriction holds.
Chapter 7

Formal Language Theory
Continued

In this chapter, we treat the context-free languages, generated with rules of the form \( A \rightarrow \psi \), where \( A \) is a non-terminal and \( \psi \) is a string of terminals and non-terminals. We also consider some higher-order systems and relations between languages.

7.1 Context-free Languages

From the definition of right-linear grammars and context-free grammars, it follows that any language that can be described in terms of a right-linear grammar can be described in terms of a context-free grammar. This is true trivially since any right-linear grammar is definitionally also a context-free grammar.

What about in the other direction though? There are languages that can be described in terms of context-free grammars that cannot be described in terms of right-linear grammars. Consider, for example, the language \( a^n b^n \): \( \{ \epsilon, ab, aabb, aaabbb, \ldots \} \). It can be generated by a context-free grammar, but not by a right-linear grammar. Here is a simple context-free grammar for this language:

\[
(7.1) \quad S \rightarrow a \; S \; b \\
S \rightarrow \epsilon
\]
Here are some sample trees produced by this grammar.

In the previous chapter, we showed how we could use the Pumping Lemma for regular languages to show that $a^n b^n$ could not be regular, and therefore cannot be described with a right-linear grammar.

Another language type that cannot be treated with a right-linear grammar is $xx^R$ where a string $x$ is followed by its mirror-image $x^R$, including strings like $aa$, $bb$, $abba$, $baaaaaab$, etc. This can be treated with a context-free grammar like this:

$$
(7.3) \quad S \rightarrow a\ S\ a \\
S \rightarrow b\ S\ b \\
S \rightarrow \epsilon
$$

This produces trees like this one:
The problem is that both sorts of language require that we keep track of a potentially infinite amount of information over the string. Context-free grammars do this by allowing the edges of the right side of rules to depend on each other (with other non-terminals in between). Thus, in a rule like \( A \rightarrow b \, C \, d \), we say that \( b \) and \( d \) depend on each other with the distance between them governed by how \( C \) can be rewritten. This sort of dependency is, of course, not possible with a right-linear grammar.

### 7.2 Pushdown Automata

Context-free grammars are equivalent to a particular simple computational model, e.g. a non-deterministic pushdown automaton (PDA). A PDA is just like a FSA, except it includes a stack, a memory store that can be utilized as each symbol is read from the tape.

The stack is restricted, however. In particular, symbols can be added to or read off of the top of the stack, but not to or from lower down in the stack. For example, if the symbols \( a, b, \) and \( c \) are put on the stack in that order, they can only be retrieved from the stack in the opposite order: \( c, b, \) and then \( a \). This is the intended sense of the term pushdown.\(^1\)

Thus, at each step of the PDA, we need to know what state we are in, what symbol is on the tape, and what symbol is on top of the stack. We can then move to a different state, reading the next symbol on the tape, adding or removing the topmost symbol of the stack, or leaving it intact. A string is accepted by a PDA if the following hold:

1. the whole input has been read;
2. the stack is empty;
3. the PDA is in a final state.

A non-deterministic pushdown automaton can be defined more formally as follows:

**Definition 14 (Non-deterministic PDA)** A non-deterministic PDA is a sextuple \( \langle K, \Sigma, \Gamma, s, F, \Delta \rangle \), where \( K \) is a finite set of states, \( \Sigma \) is a finite set (the input alphabet), \( \Gamma \) is a finite set (the stack alphabet), \( s \in K \) is the initial state, \( F \subset K \) is the set of final states, and \( \Delta \) is the control function.

\(^1\) A stack is also referred to as “last in first out” (LIFO) memory.
initial state, \( F \subseteq K \) is the set of final states, and \( \Delta \), the set of transitions is a finite subset of \( K \times (\Sigma \cup \epsilon) \times (\Gamma \cup \epsilon) \times K \times (\Gamma \cup \epsilon) \).

Let’s consider an example. Here is a PDA for \( a^n b^n \).

\[ (7.5) \quad \text{States:} \quad K = \{q_0, q_1\} \]
\[ \text{Input alphabet:} \quad \Sigma = \{a, b\} \]
\[ \text{Stack alphabet:} \quad \Gamma = \{c\} \]
\[ \text{Initial state:} \quad s = q_0 \]
\[ \text{Final states:} \quad F = \{q_0, q_1\} \]
\[ \text{Transitions:} \quad \Delta = \left\{ \begin{array}{l}
(q_0, a, \epsilon) \rightarrow (q_0, c) \\
(q_0, b, c) \rightarrow (q_1, \epsilon) \\
(q_1, b, c) \rightarrow (q_1, \epsilon)
\end{array} \right\} \]

The PDA puts the symbol \( c \) on the stack every time it reads the symbol \( a \) on the tape. As soon as it reads the symbol \( b \), it removes the topmost \( c \) from the stack and moves to state \( q_1 \), where it removes an \( c \) from the stack for every \( b \) that it reads on the tape. If the same number of instances of \( a \) and \( b \) are read, then the stack will be empty when there are no more symbols on the tape.

To see this more clearly, let us define a situation for a PDA as follows.

**Definition 15 (Situation of a PDA)** A situation of a PDA is a quadruple \((x, q, y, z)\), where \( q \in K \), \( x, y \in \Sigma^* \), and \( z \in \Gamma^* \).

The term \( x \) refers to how much of the string has been read. \( q \) is the current state. \( y \) is the remainder of the string, and \( z \) is the current contents of the stack. This is just like the situation of an FSA, except that it includes a specification of the state of the stack in \( z \).

Consider now the sequence of situations which shows the operation of the previous PDA on the string \( aaabb \).

\[ (7.6) \quad (\epsilon, q_0, aaabb, \epsilon) \vdash (a, q_0, aaabb, c) \vdash (aa, q_0, abbb, cc) \vdash \\
\quad (aaa, q_0, bbb, ccc) \vdash (aaab, q_1, bb, cc) \vdash (aaabb, q_1, b, c) \vdash \\
\quad (aaabbb, q_1, \epsilon, \epsilon) \]
The first situation \((\epsilon, q_0, aaabbb, \epsilon)\) shows that no part of the string has been read, that the PDA starts in \(q_0\), that the string to be read is \(aaabbb\), and that the stack is empty. The first three symbols are read off one by one giving rise to \((aaa, q_0, bbb, ccc)\). Here, the first three symbols have been read: \(aaa\). The PDA is still in state \(q_0\) and there are still three symbols to read: \(bbb\). Three symbols have been pushed onto the stack: \(ccc\). The next symbol is the key one. Once a \(b\) is read, the machine moves to \(q_1\) and starts pulling symbols off the stack. Once all symbols have been read, we reach \((aaabbb, q_1, \epsilon, \epsilon)\), indicating that all of \(aaabbb\) has been read, the machine is in \(q_1\), a legal final state, there is no more of the string to read, and the stack is empty. This is, therefore, a legal string in the language.

Notice that this PDA is deterministic in the sense that there is no more than one arc from any state on the same symbol.\(^2\) This PDA still qualifies as non-deterministic under Definition 14, since deterministic automata are a subset of non-deterministic automata.

The context-free languages cannot all be treated with deterministic PDAs, however. Consider the language \(xx^R\) (with \(\Sigma = \{a, b\}\)), where a string is followed by its mirror image, e.g. \(aa\), \(abba\), \(bbaabb\), etc. We’ve already seen that this is trivially generated using context-free rewrite rules. Here is a non-deterministic PDA that generates the same language.

\[(7.7)\]

\begin{align*}
\text{States:} & \quad K \quad = \quad \{q_0, q_1\} \\
\text{Input alphabet:} & \quad \Sigma \quad = \quad \{a, b\} \\
\text{Stack alphabet:} & \quad \Gamma \quad = \quad \{A, B\} \\
\text{Initial state:} & \quad s \quad = \quad q_0 \\
\text{Final states:} & \quad F \quad = \quad \{q_0, q_1\} \\
\text{Transitions:} & \quad \Delta \quad = \quad \begin{cases} 
(q_0, a, \epsilon) \rightarrow (q_0, A) \\
(q_0, b, \epsilon) \rightarrow (q_0, B) \\
(q_0, a, A) \rightarrow (q_1, \epsilon) \\
(q_0, b, B) \rightarrow (q_1, \epsilon) \\
(q_1, a, A) \rightarrow (q_1, \epsilon) \\
(q_1, b, B) \rightarrow (q_1, \epsilon) 
\end{cases}
\end{align*}

Here is the sequence of situations for \(abba\) that results in the string being

\(^2\)Notice that the PDA is not complete, as there is no arc on \(a\) from state \(q_1\).
accepted.

\[(7.8) \ (\epsilon, q_0, abba, \epsilon) \vdash (a, q_0, bba, A) \vdash (ab, q_0, ba, BA) \vdash (abb, q_1, a, A) \vdash (abba, q_1, \epsilon, \epsilon)\]

We begin in \((\epsilon, q_0, abba, \epsilon)\): none of the string has been read; we are in state \(q_0\); we have the whole string to read: \(abba\); and the stack is empty. We then read the first symbol moving to this situation: \((a, q_0, bba, A)\) Here, a single \(a\) has been read; we are still in \(q_0\); we have \(bba\) to read; and the stack has a single \(A\) in it. We next read \(b\), putting us in this situation: \((ab, q_0, ba, BA)\). Here, \(ab\) has been read; we are still in \(q_0\); we have \(ba\) yet to read; and the stack has two symbols in it now: \(BA\). The third symbol provides a choice. We can either add a \(B\) to the stack or remove a \(B\) from the stack. If we elect to remove a \(B\) from the stack, we are in this situation: \((abb, q_1, a, A)\). Three symbols have been read; we are now in \(q_1\), a single \(a\) is left to read; and the stack has a single \(A\) on it. We read the final \(a\) and move to \((abba, q_1, \epsilon, \epsilon)\). Here all symbols have been read; we are in a designated final state: \(q_1\); there are no more symbols to be read; and the stack is empty. This is therefore a legal string in the language.

Notice that at any point where two identical symbols occur in a row, the PDA can guess wrong and presume the reversal has occurred or that it has not. In the case of \(abba\), the second underlined \(b\) does signal the beginning of the reversal, but in \(abbaabba\), the second underlined \(b\) does not signal the beginning of the reversal. With a string of all identical symbols, like \(aaaaaa\), there are many ways to go wrong.

This PDA is necessarily non-deterministic. There is no way to know, locally, when the reversal begins. It then follows that the set of languages that are accepted by a deterministic PDA are not equivalent to the set of languages accepted by a non-deterministic PDA. For example, any kind of PDA can accept \(a^n b^n\), but only a non-deterministic PDA will accept \(xx^R\).

### 7.3 Equivalence of Non-deterministic PDAs and CFGs

We’ve said that non-deterministic PDAs accept the set of languages generated by context-free grammars (CFGs).
Theorem 4  Context-free grammars generate the same kinds of languages as non-deterministic pushdown automata.

This is actually rather complex to show, but we will show how to construct a non-deterministic PDA from a context-free grammar and vice versa.

7.3.1 CFG to PDA

Given a CFG $G = \langle V_N, V_T, S, R \rangle$, we construct a non-deterministic PDA as follows.

1. $K = \{q_0, q_1\}$
2. $s = q_0$
3. $F = \{q_1\}$
4. $\Sigma = V_T$
5. $\Gamma = \{V_N \cup V_T\}$

There are only two states, one being the start state and the other the sole final state. The input alphabet is identical to the set of terminal elements allowed by the CFG and the stack alphabet is identical to the set of terminal plus non-terminal elements.

The transition relation $\Delta$ is constructed as follows:

1. $(q_0, \epsilon, \epsilon) \rightarrow (q_1, S)$ is in $\Delta$.
2. For each rule of the CFG of the form $A \rightarrow \psi$, $\Delta$ includes a transition $(q_1, \epsilon, A) \rightarrow (q_1, \psi)$.
3. For each symbol $a \in V_T$, there is a transition $(q_1, a, a) \rightarrow (q_1, \epsilon)$.

The first rule above reads no symbol, but puts an $S$ on the stack. The second rule is the real workhorse. For every rule in the grammar of the form $A \rightarrow \psi$, there is a transition that pulls $A$ off the stack and puts the symbols comprising $\psi$ on the stack. Finally, for every terminal element of the grammar, there is a rule that reads that terminal from the string and pulls the corresponding element from the stack.

Let’s consider how this works with a simple context-free grammar:
(7.9)  
\[ S \rightarrow NP \ VP \]
\[ VP \rightarrow V \ NP \]
\[ NP \rightarrow N \]
\[ N \rightarrow John \]
\[ N \rightarrow Mary \]
\[ V \rightarrow \text{loves} \]

We have \( K, s, \) and \( F \) as above.

1. \( K = \{q_0, q_1\} \)
2. \( s = q_0 \)
3. \( F = \{q_1\} \)

For \( \Sigma \) and \( \Gamma \), we have:

(7.10)  \[ \Sigma = \{\text{John, loves, Mary}\} \]
\[ \Gamma = \{S, NP, VP, V, N, \text{John, loves, Mary}\} \]

The transitions of \( \Delta \) are as follows:

(7.11)  
\[(q_0, \epsilon, \epsilon) \rightarrow (q_1, S)\]
\[(q_1, \epsilon, S) \rightarrow (q_1, NP \ VP)\]
\[(q_1, \epsilon, NP) \rightarrow (q_1, N)\]
\[(q_1, \epsilon, VP) \rightarrow (q_1, V \ NP)\]
\[(q_1, \epsilon, N) \rightarrow (q_1, \text{John})\]
\[(q_1, \epsilon, N) \rightarrow (q_1, \text{Mary})\]
\[(q_1, \epsilon, V) \rightarrow (q_1, \text{loves})\]
\[(q_1, \text{John, John}) \rightarrow (q_1, \epsilon)\]
\[(q_1, \text{Mary, Mary}) \rightarrow (q_1, \epsilon)\]
\[(q_1, \text{loves, loves}) \rightarrow (q_1, \epsilon)\]
The first one corresponds to the first case in the construction. The next six correspond to individual rewrite rules and the second case in the construction. The last three above correspond to each of the terminal elements of the grammar and the third case of the construction.

Let’s now look at how this PDA treats an input sentence like Mary loves John. Following, we give the sequence of situations that result in the sentence being accepted.

\[
\begin{align*}
(7.12) & \quad (\epsilon, q_0, \text{Mary loves John}, \epsilon) \vdash \\
& \quad (\epsilon, q_1, \text{Mary loves John}, S) \vdash \\
& \quad (\epsilon, q_1, \text{Mary loves John}, NP \ VP) \vdash \\
& \quad (\epsilon, q_1, \text{Mary loves John}, N \ VP) \vdash \\
& \quad (\epsilon, q_1, \text{Mary loves John}, Mary \ VP) \vdash \\
& \quad (\text{Mary}, q_1, \text{loves John}, VP) \vdash \\
& \quad (\text{Mary}, q_1, \text{loves John}, V \ NP) \vdash \\
& \quad (\text{Mary}, q_1, \text{loves John}, \text{loves} \ NP) \vdash \\
& \quad (\text{Mary loves}, q_1, \text{John}, NP) \vdash \\
& \quad (\text{Mary loves}, q_1, \text{John}, N) \vdash \\
& \quad (\text{Mary loves}, q_1, \text{John}, \text{John}) \vdash \\
& \quad (\text{Mary loves John}, q_1, \epsilon, \epsilon)
\end{align*}
\]

First, the non-terminal elements are put onto the stack. We then proceed to expand each non-terminal down to terminals in a left-to-right fashion, replacing symbols on the stack as we expand and popping symbols off the stack as we reach terminal elements. Finally, the string is completely read and the stack is empty.

This is not a proof that an equivalent non-deterministic PDA can be constructed from any CFG, but it shows the basic logic of that proof.

7.3.2 PDA to CFG

In this section, we show how to construct a context-free grammar from a pushdown automaton. The construction is a little tricky as we construct non-terminals from state/stack sequences of the form \([p, X, q]\), where \(p\) and
q are states in the PDA and X is a symbol in the stack alphabet Γ.\footnote{See Hopcroft and Ullman (1979) for details.}

There are two parts to the construction:

**Initial Rule** Add rewrite rules of the form $S \rightarrow [q_0, \epsilon, p]$ for all states $p \in Q$.

**Transition Rule** Add rules of the form

\[
[q, A, q_{m+1}] \rightarrow a \ [q_1, B_1, q_2] \ [q_2, B_2, q_3] \ \ldots \ [q_m, B_m, q_{m+1}]
\]

for each $q, q_1, q_2, \ldots, q_{m+1}$ in $Q$, each $a$ in $\Sigma \cup \{\epsilon\}$, and $A, B_1, B_2, \ldots, B_m$ in $\Gamma$, such that $\delta(q, a, A)$ contains $(q_1, B_1B_2\ldots B_m)$. (If $m = 0$, then the production is $[q, A, q_1] \rightarrow a$.)

It’s best to understand the Transition Rule as applying in three cases: removing a symbol from the stack, changing a symbol on the stack, and adding a symbol to the stack. Let’s take these in order of complexity. If a symbol $A$ is removed from the stack when reading symbol $a$ and we move from state $q$ to state $q_1$, then we add a rule of the form $[q, A, q_1] \rightarrow a$.

If we replace the topmost symbol $A$ on the stack with another (singleton) symbol $B$, reading the symbol $a$, and moving from state $q$ to state $q_1$, then we add rules of this form to the grammar: $[q, A, p] \rightarrow a \ [q_1, B, p]$. We add as many instances of this rule as there are states in the original PDA, with $p$ being replaced by each of those state names.

Finally, if we *add* a symbol to the stack, effectively replacing $A$ with $BA$, while reading $a$ and moving from state $q$ to $q_1$, then we add rules of the form: $[q, A, r] \rightarrow a \ [q_1, B, p] \ [p, A, r]$, for every possible combination of states $p$ and $r$ in the original PDA. Thus, if the original PDA had three states, we would add *nine* rewrite rules.

Notice that this requires that we keep track of what symbols might be on the top of the stack at any time.

Let’s show how this works for the PDA for $a^n b^n$ in (7.5). The Initial Rule gives us two rules:

\[
(7.13) \quad S \rightarrow [q_0, \epsilon, q_0] \\
S \rightarrow [q_0, \epsilon, q_1]
\]

As usual, $S$ is the designated start symbol for the grammar. The symbols on the right side are constructed from the start state for the PDA $q_0$ and the set of all possible states in the automaton: $\{q_0, q_1\}$. 

To apply the Transition Rule, we must consider the transitions of the original PDA, repeated below.

\[(7.14) \quad (q_0, a, \epsilon) \rightarrow (q_0, c) \]
\[(q_0, b, c) \rightarrow (q_1, \epsilon) \]
\[(q_1, b, c) \rightarrow (q_1, \epsilon) \]

The latter two cases are straightforward. They involve popping off a symbol from the stack and thus entail adding one rewrite rule each.

\[(7.15) \quad [q_0, c, q_1] \rightarrow b \]
\[ [q_1, c, q_1] \rightarrow b \]

The first case is more complex, as it really stands in for two different situations: the stack might be empty or the stack might already have a c on it. These two cases are given below:

\[(7.16) \quad (q_0, a, \epsilon) \rightarrow (q_0, c) \]
\[(q_0, a, c) \rightarrow (q_0, cc) \]

For the first case, we add two rules:

\[(7.17) \quad [q_0, \epsilon, q_0] \rightarrow a \quad [q_0, c, q_0] \]
\[ [q_0, \epsilon, q_1] \rightarrow a \quad [q_0, c, q_1] \]

For the second case, since it involves the addition of a stack symbol, we add four rules:

\[(7.18) \quad [q_0, c, q_0] \rightarrow a \quad [q_0, c, q_0] \quad [q_0, c, q_0] \]
\[ [q_0, c, q_0] \rightarrow a \quad [q_0, c, q_1] \quad [q_1, c, q_0] \]
\[ [q_0, c, q_1] \rightarrow a \quad [q_0, c, q_0] \quad [q_0, c, q_1] \]
\[ [q_0, c, q_1] \rightarrow a \quad [q_0, c, q_1] \quad [q_1, c, q_1] \]

We collect all the rules of the grammar below:
(7.19) \[ S \rightarrow [q_0, \epsilon, q_0] \]
\[ S \rightarrow [q_0, \epsilon, q_1] \]
\[ [q_0, c, q_1] \rightarrow b \]
\[ [q_1, c, q_1] \rightarrow b \]
\[ [q_0, \epsilon, q_0] \rightarrow a [q_0, c, q_0] \]
\[ [q_0, \epsilon, q_1] \rightarrow a [q_0, c, q_1] \]
\[ [q_0, c, q_0] \rightarrow a [q_0, c, q_0] [q_0, c, q_0] \]
\[ [q_0, c, q_0] \rightarrow a [q_0, c, q_1] [q_1, c, q_0] \]
\[ [q_0, c, q_1] \rightarrow a [q_0, c, q_0] [q_0, c, q_1] \]
\[ [q_0, c, q_1] \rightarrow a [q_0, c, q_1] [q_1, c, q_1] \]

This produces a tree like the following for \( aabb \).

(7.20)
\[
\begin{array}{c}
S \\
| \\
[q_0, \epsilon, q_1] \\
| \\
a [q_0, c, q_1] \\
| \\
a [q_0, c, q_1] [q_1, c, q_1] \\
| b \\
| b
\end{array}
\]

This is not a rigorous proof that an equivalent CFG can be constructed from any PDA, but it shows the general character of that proof.

### 7.4 Closure Properties of Context-free Languages

The context-free languages are closed under a number of operations including union, concatenation, and Kleene star. They are \textit{not} closed under comple-
mentation, nor are they closed under intersection.\footnote{Though they are, as you might expect, closed under intersection with a regular language.}

The demonstration that they are not generally closed under intersection is easy to see. One can show that $a^n b^n c^n$ is beyond the limits of context-free grammar. Now we know that $a^n b^n$ is context-free. We can complicate that just a little and still stay within the limits of context-free grammar: $a^n b^n c^m$, where though the first two symbols must be paired, there can be any number of instances of the third. If we try to intersect $a^n b^n c^m$ and $a^m b^n c^n$, which is also of course describable with a context-free grammar, we get $a^n b^m c^n$, which is not context-free.

### 7.5 Pumping Lemma for Context-free Languages

The Pumping Lemma also applies to context-free languages. It is similarly opaque, but we can try to make some sense of it. The basic idea is to track down how context-free languages allow for infinity.

If we think of the context-free languages in terms of a PDA, the basic mechanism for infinity is the stack, allowing an infinite number of elements to be stored and then read back. This allows for two matched infinitely long substrings in a word of the language. One substring is where the stack symbols are added and the other is where the stack symbols are read back.

Another way to think of this is in terms of a context-free grammar. Infinity comes about because of recursion. Some rule either feeds itself, e.g. $A \rightarrow a A a$ or there are several rules that create a similar cycle, e.g. $A \rightarrow a B$ and $B \rightarrow A a$. Either way, we get an infinite number of words in the language. Focusing for the moment on the simple case, the key is that when a rule is recursive, it allows material to each side of the repeated symbol to recur. Thus in $A \rightarrow a A a$, there is a symbol to each side of the $A$ on the right side of the rule that can be repeated. Therefore, there are, once again, two infinitely long substrings in an infinite context-free language.

Here is a formal statement of the Pumping Lemma:\footnote{As in the previous chapter, we give this as a theorem, but refer to it as a lemma to keep with the traditional nomenclature.}

**Theorem 5** If $L$ is a context-free language, then there is a constant $n$, such that...
that if $z$ is in $L$ and $|z| \geq n$, then we may write $z = uvwxy$ such that: 
$|vx| \geq 1$, $|vwx| \leq n$, and for all $i \geq 0$, $uv^iwx^iy$ is in $L$.

The basic idea is very similar to the Pumping Lemma for Regular Languages. Once the strings of a language reach a certain point in length, then we are always able to identify two substrings that can be repeated any number of times to produce infinitely more strings in the language.

Consider, for example, the language $a^n b^n$, which we know is context-free. Here, $n = 2$. That is, once we have a string $ab$, we can identify all the variables of $uvwxy$, i.e. $u = \epsilon$, $v = a$, $w = \epsilon$, $x = b$, and $y = \epsilon$. For example if $i = 3$, then we have a legal string: $uv^3wx^3y = aaabbb$.

The Pumping Lemma for Context-free Languages can be used to show that some languages are not context-free. For example, the language $a^n b^n c^n$ exceeds context-free. This language allows strings like:

(7.21) $\{abc, aabbcc, aaabbbccc, \ldots\}$

Imagine that we set $n$ to 3. If so, then we must be able to find two repeatable substrings in any word of three or more symbols. Take the string $abc$. If we require that $|vx| \geq 1$ and $|vwx| \leq 3$, then only certain assignments are possible and all of them make incorrect predictions when $i = 2$. These are all diagrammed in the following chart.\(^6\)

\(^6\)The chart omits cases where a symbol can be assigned to more than one of $u$, $v$, or $y$. 

The first five columns show the assignments and the rightmost column shows what happens when $i = 2$. The same result obtains for any value of $n$ and $a^n b^n c^n$ is not context-free.

### 7.6 Natural Language Syntax

The syntax of English cannot be regular. Consider these examples:

(7.23) The cat died.
     The cat [the dog chased] died.
     The cat [the dog [the rat bit] chased] died.
     ;

---

<table>
<thead>
<tr>
<th>(7.22)</th>
<th>$u$</th>
<th>$v^i$</th>
<th>$w$</th>
<th>$x^i$</th>
<th>$y$</th>
<th>$i = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ab$</td>
<td>$c$</td>
<td>$e$</td>
<td>$e$</td>
<td>$e$</td>
<td>$abcc$</td>
<td></td>
</tr>
<tr>
<td>$ab$</td>
<td>$e$</td>
<td>$e$</td>
<td>$c$</td>
<td>$e$</td>
<td>$abcc$</td>
<td></td>
</tr>
<tr>
<td>$ab$</td>
<td>$e$</td>
<td>$e$</td>
<td>$b$</td>
<td>$c$</td>
<td>$abbc$</td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>$bc$</td>
<td>$b$</td>
<td>$c$</td>
<td>$e$</td>
<td>$abc$</td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$e$</td>
<td>$e$</td>
<td>$abc$</td>
<td></td>
</tr>
</tbody>
</table>
This is referred to as center embedding. Center-embedded sentences generally require a match between the number of subjects and the number of verbs. These elements do not occur adjacent to each other (except for the most embedded pair). This, then, is equivalent to $a^n b^n$ and beyond the range of the regular languages.\footnote{This argument is due to Partee et al. (1990).}

Most speakers of English find sentences like these rather marginal once they get above two or three clauses. It is thus a little disturbing that the best example of how natural language syntax cannot be regular is of this sort. The problem can be made clear with a formal example. We know that $a^n b^n$ is context-free. What if we bound $n$? For example, imagine we have a language $L = \{ab, aabb, aaabbb\}$. Even though the pattern is the same, here the language is finite and can be defined by simply enumerating its members. Hence, such a language is certainly no more than regular.

If we assume that center-embedded sentences are grammatical up to infinity, is natural language syntax context-free or context-sensitive? Shieber (1985) argues that natural language syntax must be context-sensitive based on data from Swiss German. Examples like the following are grammatical. (Assume the sentence begins with the phrase "Jan säit das ‘John said that’.")

\begin{equation}
\text{mer d’chind em Hans es huus lönd hälfe aastiiche.} \\
\text{we children Hans house let help paint} \\
\text{‘…we let the children help Hans paint the house.’}
\end{equation}

This is equivalent to the language $ww$, e.g. $aa$, $abab$, $abbabb$, etc., which is known not to be context-free.\footnote{Shieber actually goes further and shows that examples of this sort are not simply an accidental subset of the set of Swiss German sentences, but I leave this aside here.}

If the Swiss German pattern is correct, then it means that any formal account of natural language syntax requires more than a PDA and that a formalism based on context-free grammar is inadequate. Notice, however, that, once again, it is essential that the embedding be unbounded. That is, while $ww$ is clearly beyond context-free, $\{aa, abab, abcabc\}$ is finite.

### 7.7 Determinism

Since at least some context-free languages require non-deterministic PDAs, and non-deterministic PDAs cannot be reduced to deterministic ones, deter-
mining whether some particular string is accepted by such a PDA is a more complex operation than for the regular languages.

Recall that, for a regular language, we can always construct a DFA. Thus, for the regular languages, the number of steps we need to consider to determine the acceptability of some string \( s \) is equal to the length of that string.

On the other hand, if we are interested in whether some string \( s \) is accepted by a non-deterministic PDA, we must keep considering paths through the automaton until we find one that terminates with the appropriate properties: end of string, empty stack, in a final state. This may be quite a few paths to consider. Recall the context-free language \( xx^R \) and the non-deterministic PDA we described to treat it. For any string of length \( n \), we must entertain the hypothesis that the reversal begins at any point between 1 and \( n \). This entails that we must consider lots of paths for a long string.\(^9\)

What this means, in concrete terms, is that if some phenomenon can be treated in finite-state terms or in context-free terms, and efficiency is a concern, go with the finite-state treatment.

### 7.8 Other Machines

There are other machines far more powerful than PDAs. For example, there are *Turing machines* (TMs). These are like FSAs except i) the reading head can move in either direction, and ii) it can write to as well as read from the tape. These properties allow TMs to use empty portions of the input tape as an unbounded memory store without the access limitations of a stack. Formally, a TM is defined as follows.

**Definition 16 (Turing Machine)** A *Turing machine* (TM) is a quadruple \((K, \Sigma, q_0, \delta)\), where \( K \) is a finite set of states, \( \Sigma \) is a finite alphabet including \#, \( q_0 \in K \) is the start state, and \( \delta \) is a partial function from \( K \times \Sigma \) to \( K \times (\Sigma \cup \{L,R\}) \).

Here \# is used to mark initially empty portions of the input tape. The logic of \( \delta \) is that it maps from particular state/symbol combinations to a new

\(^9\)It might be thought that we must consider an infinite number of paths, but this is not necessarily so. Any non-deterministic PDA with an infinite number of paths for some finite string can be converted into a non-deterministic PDA with only a finite number of paths for some finite string. See Hopcroft and Ullman (1979). The effect is that for a string \( s \) of length \( n \) and a non-deterministic automaton with \( m \) states, we may have to consider as many as \( n^m \) paths.
state, simultaneously (potentially) writing a symbol to the tape, or moving left or right.

TMs can describe far more complex languages than are thought to be appropriate for human language. For example $a^n b^n$ can be treated with a TM. Likewise, $a^n$, where $n$ is prime, can be handled with a TM. Hence, while there is a lot of work in computer science on the properties of TMs, there has not been a lot of work in grammatical theory using them.

Another machine type that we have not treated so far is the finite state transducer (FST). The basic idea behind a transducer is that we start with an FSA, but label the arcs with pairs of symbols. The machine can be thought of as reading two tapes in parallel. If the pairs of symbols on each respective arc match what is on the two tapes—and the machine finishes in a designated final state—then the pair of strings is accepted. Another way to think of these, however, is that the machine reads one tape and spits out some symbol every time it transits an arc (perhaps writing those latter symbols to a new blank tape). Thus, if an arc is labeled \(a : b\), the machine would read an \(a\) and spit out a \(b\).

Formally, we can define an FST as follows:

**Definition 17 (FST)** An FST is a sextuple \((K, \Sigma, \Gamma, s, F, \Delta)\) where \(K\) is a finite set of states, \(\Sigma\) is the finite input alphabet, \(\Gamma\) is the finite output alphabet, \(s \in K\) is the start state, \(F \subseteq K\) is the set of final states and \(\Delta\) is a relation from \(K \times (\Sigma \cup \epsilon)\) to \(K \times (\Gamma \cup \epsilon)\).

The relation \(\Delta\) moves from state to state pairing symbols of \(\Sigma\) with symbols of \(\Gamma\). The instances of \(\epsilon\) in \(\Delta\) allow it to insert or remove symbols, thus matching strings of different lengths.

For example, here is an FST that operates with the alphabet \(\Sigma = \{a, b, c\}\), where anytime a \(b\) is confronted on one tape, the machine spits out \(c\).

\[
\begin{array}{cc}
\text{b : c} & \text{c : c} \\
\text{a : b} & \text{a : b}
\end{array}
\]

Such an FST would convert \(abcbcaa\) into \(acccccaa\).

The interest of such machines is twofold. First, like FSAs they are quite restricted in power and very well understood. Second, many domains of lan-
guage and linguistics are modeled with input–output pairings and a transducer provides a tempting model for such a system. For example, in phonology, if we posit rules or constraints that nasalize vowels before nasal consonants, we might model that with a transducer that pairs oral vowels with nasal vowels just in case the following segment is a nasal consonant.

Let’s look a little more closely at the kinds of relationships multi-tape transducers allow (Kaplan and Kay, 1994). First we need a notion of n-way concatenation. This generalizes the usual notion of concatenation to transducers with more than one tape.10

**Definition 18 (n-way concatenation)** If \( X \) is an ordered tuple of strings \( \langle x_1, x_2, \ldots, x_n \rangle \) and \( Y \) is an ordered tuple of strings \( \langle y_1, y_2, \ldots, y_n \rangle \) then the n-way concatenation of \( X \) and \( Y \), \( X \cdot Y \) is defined as \( \langle x_1 y_1, x_2 y_2, \ldots, x_n y_n \rangle \).

It’s also convenient to have a way to talk about alphabets that include \( \epsilon \). We define \( \Sigma^\epsilon = \{ \Sigma \cup \epsilon \} \). With these in place, we can define the notion of a regular relation (Kaplan and Kay, 1994).

**Definition 19 (Regular Relation)** We define the regular relations as follows:

1. The empty set and all \( a \) in \( \Sigma^\epsilon \times \ldots \times \Sigma^\epsilon \) are n-way regular relations.

2. If \( R_1 \), \( R_2 \), and \( R \) are all regular relations, then so are:
   - \( R_1 \cdot R_2 = \{ xy \mid x \in R_1, y \in R_2 \} \) (n-way concatenation)
   - \( R_1 \cup R_2 \) (union)
   - \( R^* = \bigcup_{i=0}^{\infty} R^i \) (n-way Kleene closure)

3. There are no other regular relations.

First, we have the base case that any combination of symbols is a regular relation. For example, \( a : b \) is a regular relation as is \( b : c \), or \( \epsilon : a \) or \( b : b \). There are then three recursive clauses. The first allows us to concatenate, e.g. \( (a : b) \cdot (b : c) = (ab) : (bc) \). The second allows us to take the union of two regular relations, e.g. \( (a : b) \cup (b : c) \). Finally, the third recursive clause allows us to repeat a relation any number of times, e.g. \( (a : b)^* \). Notice how

---

10The formal definitions here are general and allow for two or more tapes. Most applications in language only involve two tapes.
these parts of the definition are exactly parallel to the parts of the definition of a regular language (Definition 13 on page 112).

We can show that the regular relations are equivalent to finite state transducers. There are two parts to such a demonstration. First, we would need to show that any regular relation can be mapped to an FST. Then we would need to show that every FST can be mapped to a regular relation.

Let’s consider this from the perspective of creating a regular relation from an FST, e.g. the one in (7.25). There are three simple relations indicated in the FST.

\[
\{(a : a), (b : c), (c : c)\}
\]

Each loop back to the start state can be seen as a Kleene star applied to a simple regular relation. Thus, we have:

\[
\{(a : a)^*, (b : c)^*, (c : c)^*\}
\]

Finally, multiple outgoing arcs from a single node express union, so this can be converted to:

\[
(a : a)^* \cup (b : c)^* \cup (c : c)^*
\]

Since each step corresponds to a clause in the definition of the regular relations, it follows that we can construct a regular relation from an FST.\(^\text{11}\)

It should be apparent that the regular relations are closed under the usual regular operations.\(^\text{12}\) The regular relations are closed under a number of other operations too, e.g. the ones above, but also reversal, inverse, and composition. Finally, it should be apparent that each half of a regular relation is itself a regular language. For example, in the case above, \((a : a)^* \cup (b : c)^* \cup (c : c)^*\), we have \((a^*|b^*|c^*)\) as the upper language and \((a^*|c^*|c^*) = (a^*|c^*)\) as the lower language, both of which are regular.

They are not closed under intersection and complementation, however. For example, imagine we have

\(^{11}\)We can do this in the other direction as well, but we leave that as an exercise.

\(^{12}\)Note that the regular relations are equivalent to the “rational relations” of algebra for the mathematically inclined.
\( R_1 = \{(a^n) : (b^n c^n) \mid n \geq 0\} \)

and

\( R_2 = \{(a^n) : (b^n c^n) \mid n \geq 0\} \)

Each of these is itself a regular relation. For example, we can show that \((a^n) : (b^n c^n)\) is built up according to the definition of regular relations.

\[(7.29)\]

\[
\begin{align*}
(a : b) & : \text{base} \\
(a : b)^* & : \text{clause 3} \\
\epsilon : c & : \text{base} \\
(\epsilon : c)^* & : \text{clause 3} \\
(a : b)^* \cdot (\epsilon : c)^* & : \text{clause 1}
\end{align*}
\]

The number of instances of \(a\) will match the number of instances of \(b\), but the number of instances of \(c\) will be independent. A similar derivation can be given for \(R_2\).

The intersection is clearly not regular. The intersection of \(R_1\) and \(R_2\) would be \((a^n) : (b^n c^n)\). The lower language here would be \(a^n b^n\) and we know that is regular. Hence, since the upper and lower languages of a regular relation are themselves regular and \(a^n b^n\) is not regular, it follows that \((a^n) : (b^n c^n)\) is not regular that the regular relations are not closed under intersection.

It then follows, of course, that the regular relations are not closed under complementation (by DeMorgan’s Law).\(^\dagger\)

### 7.9 Summary

The chapter began with a discussion of the context-free languages showing how they can be accommodated by non-deterministic pushdown automata. We showed how deterministic pushdown automata were not sufficient to accommodate all the context-free languages.

\(^\dagger\)Same-length regular relations are closed under intersection.
We went on to discuss the closure properties of this class of languages and the Pumping Lemma for Context-free Languages.

We considered the implications of formal language theory for natural language syntax, citing several arguments that natural language must be context-free and may even be context-sensitive.

Last, we briefly reviewed two additional abstract machine types: Turing machines and finite state transducers. We discussed the regular relations and their equivalence to transducers.

### 7.10 Exercises

1. In the PDA in (7.7), why must the string have two identical symbols in a row for it to be possible to guess wrong?

2. Explain why $ww$ cannot be context-free.

3. Assuming the alphabet $\{a, b, c\}$, write a transducer which replaces any instance of $a$ that precedes a $b$ with an $c$. Otherwise, strings are unchanged.

4. Start with the context-free grammar given for $xxR$ in the chapter and show how a PDA can be constructed for it using the procedure in the chapter.

5. Start with the pushdown automaton given for $xxR$ in the chapter and show how a CFG can be constructed for it using the construction in the chapter.

6. Give a finite state transducer that models the phonology of nasalization in English: vowels are nasalized before nasal consonants. Assume $\Sigma = \{V, N, C\}$ and $\Gamma = \{V, N, C, E\}$, where $V$ is an oral vowel, $N$ is a nasal consonant, $C$ is an oral consonant, and $E$ is a nasal vowel.

7. Construct a finite state transducer for this regular relation: $(((a : a) \cdot (a : a)) \cup (b : c)^*) \cdot (b : b)$.

8. Explain why the regular relations are closed under the reverse operation.

9. Are there any instances of $ww$ in human language?
10. Find a language phenomenon that we have not discussed and show what level of grammar is required to describe it.
Chapter 8

Probability

In this section, we go over some simple concepts from probability theory. We integrate these with ideas from formal language theory in the next chapter.

8.1 What is it and why should we care?

Probability theory is a conceptual and mathematical treatment of likelihood. The basic idea is that, in some domain, events may not occur with certainty and probability allows us to treat such events in a formal and precise fashion.

We might think that linguistic events are all certain. That is, theories of language are not concerned with judgments or intuitions as events that occur with something less than 100% certainty. For example, we treat the grammaticality or ungrammaticality of any particular sentence as being certain. Thus John loves Mary is completely grammatical and loves John Mary is completely ungrammatical.

Less clear judgments are usually ascribed to performance factors. For example, the center-embedded constructions we considered in the previous chapter arguably exhibit gradient acceptability, rather than grammaticality, at least on orthodox assumptions. Acceptability differences are taken to be a consequence of performance, while grammaticality differences are taken to be a consequence of grammar.

There are, however, a number of places where probability does play a role in language. Consider first typological distribution, the range of patterns that occur in the languages we know about. There are presumably an infinite number of possible languages, yet the set of extant described languages is
only a finite subset of those. If a pattern occurs in that subset, do we have reason to assume that it also holds of the entire infinite set? For example, all extant languages have something like the vowel [a]. For example, there is no language where the word for ‘apple’ is *fpststsatststsf*. Are these representative of the set of possible languages? Can we conclude that any language must have the vowel [a] and that no language can call an apple *fpststsatststsf*?

Likewise there are clear skewings among the extant set of languages, cases that appear exceptional given the available sample of languages. Swiss German alone exhibits cross-serial dependencies. Dyirbal is syntactically ergative, perhaps the only such system.\(^1\) Hixkaryana is VOS, again perhaps the only system of this sort.\(^2\) Are these really exceptional systems?

A similar problem arises due to electronic corpora and the generalizations one might conclude from them. New technology has made huge corpora available to everyone, but this leads to statistical questions. For example, if you find no examples of some particular construction in a Google query, does that mean it doesn’t occur?

A third reason for know about this is that sometimes statistical effects have direct consequences in the grammar. For example, Fidelholtz (1975) shows that vowel reduction in English correlates with lexical frequency.

Vowel reduction refers to the fact that certain vowels can be pronounced as a shorter “laxer” schwa when unstressed. For example, the first vowel of the word *parent* [pær@nt] is reduced when stress moves from the first vowel to the second in a form like *parental* [pær@nt@l].

Thus, a relatively frequent word like *astronomy* undergoes initial reduction more readily than a less frequent word like *gastronomy*. In other words, a pronunciation [astr@nomic] with a reduced first vowel is far more likely than [astr@nomic] with a full first vowel. On the other hand, [g@str@nomic] with a reduced first vowel is less likely than [g@str@nomic] with a full first vowel. According to Fidelholtz, this difference in pronunciation follows from the different frequencies or likelihoods of the words: more frequent or more likely words undergo this initial reduction more readily.

Similarly, Hooper (1976) shows that more frequent words undergo medial syncope more readily. Medial syncope refers to a phenomenon whereby—in certain configurations—a medial vowel is not pronounced. Compare the

---

\(^1\) Syntactic ergativity refers to a situation where the subjects of verbs with no objects exhibit similar behavior to the objects of verbs. This can be opposed to languages like English where objects behave differently from subjects generally.

\(^2\) VOS refers to the unmarked word order in sentences: verb – object – subject.
pronunciations of *opera* [ápra] and *operatic* [ápəˈrætɪk]. In the first word, there is no vowel pronounced between the [p] and the [r]. In the second word, there is. The configurations in which this occurs are rather complex and not relevant here (Hammond, 1997). What Hooper shows is that more frequent or likely words undergo this process more readily. For example, we find relatively frequent *memory* [məˈmɛri] with syncope, but relatively infrequent *mammary* [mæˈmərɪ] without.

All of these facts suggest that it might be reasonable to incorporate a notion of likelihood into the theory of language.

### 8.2 Basic Probability

In this section we define and explain several basic mathematical notions of probability.

First, we can define the probability of an event $e$ as:

\[
p(e) = \frac{n}{N}
\]

where $n$ is the number of outcomes that qualify as $e$ and $N$ is the total possible number of outcomes. For example, the probability of throwing a three with a single die is $\frac{1}{6}$. Here there is a single outcome consistent with $e$, and six possible outcomes in total. We can consider more complex cases too. The probability of throwing a number less than three is $\frac{2}{6} = \frac{1}{3}$.

To use an example involving language, we might consider the probability that the next word out of my mouth will be *probability*. If, for the sake of discussion, we say that English has 19528 words and all words are equally likely, then the probability of this event is approximately $\frac{1}{19528} = .00005$.\(^3\) If, instead, we were interested in the probability that the next word out of my mouth begins with the sound [p] and there are 1704 words with this property and all words are equally likely, then the probability of this event would be $\frac{1704}{19528} = .08726$.

Several important properties follow from this simple definition. First, it follows that if an event is impossible, it will have a probability of zero.

---

\(^3\)English, of course, has many more words than this, but this particular number is chosen from a convenient database (available from the author’s website) of English words used to generate all the language examples in this chapter.
For example, the probability of throwing a seven with a single die is $\frac{0}{6} = 0$. The probability that the next word out of my mouth is [karandaš] is $\frac{0}{100000}$.\footnote{The Russian word for ‘pencil’}. Likewise, the probability of a certain or sure event is one. Thus the probability of throwing a number less than seven with a single die is $\frac{6}{6} = 1$. Analogously, the likelihood that the next word out of my mouth will be a word of English is $\frac{19528}{19528} = 1$. Finally, it follows as well that probability values range between 0 and 1.

It should also now be clear that the probability of a disjunction of independent events is the sum of their probabilities.

\begin{equation}
(p_1 \lor p_2) = p_1 + p_2
\end{equation}

For example, the probability of throwing a one or a two is the probability of a one plus the probability of a two.

\[ p(1 \lor 2) = p(1) + p(2) = \frac{1}{6} + \frac{1}{6} = \frac{2}{6} = \frac{1}{3} \]

Similarly, the probability of saying the word probability ($e_1$) or the word likelihood ($e_2$) is the sum of their independent probabilities, presumably:

\[ \frac{1}{19528} + \frac{1}{19528} = \frac{2}{19528} = .0001 \]

Another way to look at this is to see $e_1$ and $e_2$ each as sets of outcomes. We then calculate the disjunction of those events by calculating the probability over the union of their component events, i.e. $p(e_1 \lor e_2) = p(e_1 \cup e_2)$. The examples we have given so far involve events that are constituted by only a single outcome, but the principle applies as well to cases where an event is constituted by multiple outcomes.

For example, what is the probability of throwing a three or throwing an even number? The probability of throwing a three is one in six and the event is constituted by a single outcome: $\frac{1}{6}$. The probability of throwing an even number is three in six and there are three outcomes in the event: $\frac{3}{6}$. The probability of the disjunction can be calculated by adding together the independent probabilities:

\[ \frac{1}{6} + \frac{3}{6} = \frac{4}{6} = \frac{2}{3} \]
or by unioning together the independent events and by taking the probability of the new event:

\[
p(\{3\} \cup \{2, 4, 6\}) = p(\{2, 3, 4, 6\}) = \frac{4}{6}
\]

The same reasoning would apply to finding the probability of uttering a word that begins with \([p]\) or the word likelihood. There are 1704 outcomes consistent with the first event and one separate outcome consistent with the second. Hence, the probability of one or the other occurring is:

\[
\frac{1705}{19528} = .08731
\]

This generalizes to the disjunction of overlapping events. Imagine we are interested in the probability of throwing a number less than three \(e_1\) or an even number \(e_2\). The first event has these outcomes: \(\{1, 2\}\), and the second has these: \(\{2, 4, 6\}\); the set of outcomes overlaps. Their union is: \(\{1, 2, 4, 6\}\), so the probability of their disjunction is \(p(e_1 \vee e_2) = \frac{4}{6} = \frac{2}{3}\).

A rather trivial language example would be the probability of uttering a word that begins with \([p]\) or the word probability. We have assumed that there are 1704 words that begin with \([p]\), but probability is one of those. Hence the probability of this disjunction is the union of their sets of outcomes: \(\frac{1704}{19528}\).

The conjunction of overlapping events—that both events occur—is also straightforward. It is the simply the number of cases compatible with both events: the intersection of the outcomes that make up those events. For example, the probability of throwing a number greater than two and less than five, is the intersection of the cases compatible with each. Thus, the cases compatible with a roll greater than two are \(\{3, 4, 5, 6\}\). The cases compatible with a role less than five are \(\{1, 2, 3, 4\}\). The intersection of these is \(\{3, 4\}\), so the probability of both occurring is \(\frac{2}{6} = \frac{1}{3}\).

The probability of uttering a word that both begins with \([p]\) and ends with \([p]\) is found by finding all the words that begin with \([p]\) and all the
words that end with [p] and then finding the intersection of these sets: the set of words that satisfy both properties.\(^5\)

It also follows that the combined probabilities of all independent events is one. For a single die, the set of all possible events is \{1, 2, 3, 4, 5, 6\}. The probability of each is \(\frac{1}{6}\), and their combined probability is:

\[
\frac{1}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = 1
\]

In a similar fashion, the probability of a word beginning with any of [a], [b], [c], [d], etc. is 1. Whatever the likelihood of each letter is as the first letter of a word, if we combine all possible choices, it must sum to 1.

This is also true of more complex events. For example, the probability of throwing a number less than three is \(\frac{2}{6}\) and the probability of throwing a number three or greater is \(\frac{4}{6}\). (These two events exhaust the event space.) Their combined probability is then \(\frac{2}{6} + \frac{4}{6} = 1\).

A language example would be the probability of a word beginning with a vowel plus the probability of a word beginning with a consonant. Let’s assume that all letters are equally likely and that there are only five vowel letters. The probability that a word begins with a vowel is then \(\frac{5}{26}\) and the probability that a word begins with a consonant is \(\frac{21}{26} = .808\). These sum as expected: \(.192 + .808 = 1\).\(^6\)

It now also follows that the probability of some event \(e\) not occurring is \(1 - p(e)\). For example, if the probability of throwing a number less than two is \(\frac{1}{6}\), and the combined probability of all possible outcomes is one, then the probability of not throwing a number less than two is \(1 - \frac{1}{6} = \frac{5}{6}\). Returning to our initial letter example, it follows that the probability of a word beginning with a consonant is 1 minus the probability that a word begins with a vowel: \(1 - .192 = .808\).

\(^5\)In the sample used to generate the examples of this chapter, those figures are as follows.

<table>
<thead>
<tr>
<th>Words that begin with [p]</th>
<th>1704</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words that end with [p]</td>
<td>338</td>
</tr>
<tr>
<td>Words that begin and end with [p]</td>
<td>25</td>
</tr>
</tbody>
</table>

\(^6\)The actual facts are more complex. There are more than five vowels and twenty-one consonants; these segments are not equally likely; and spelling only indirectly reflects the actual pronunciation.
8.3 Combinatorics

Let’s now consider how to “count cases”. Let’s consider first permutations, the possible ways of ordering some set of elements. If we have \( n \) elements, then there are \( n! \) ways of ordering those elements. For example, if we have three elements \( \{a, b, c\} \), then there are \( 3! = 3 \times 2 \times 1 = 6 \) ways of ordering those elements:

\[
(8.3) \quad a > b > c \quad a > c > b \quad b > a > c \\
\quad b > c > a \quad c > a > b \quad c > b > a
\]

We can make intuitive sense of this easily. If we are attempting to order three objects in all possible ways, we have three ways to choose a first object. Once we’ve done that, we then have two ways to choose a second object. Once we’ve done that, we only have a single way to choose a third object. This gives us: \( 3 \times 2 \times 1 \).

Let’s move back to the dice example. How many options are there for drawing a card off the top of a full deck of cards? 52. How many ways are there to draw three cards? There are 52 ways to draw a first card and then 51 ways to draw a second card and 50 ways to draw a third. Each choice is independent, once the previous card has been selected, but the number of available cards is reduced by on each time, so we have:

\[
52 \times 51 \times 50
\]

A similar example in a language domain might be how many ways are there to construct a three-letter word with three different letters? Given 26 possible letters, there are 26 ways to choose a first letter, 25 ways to choose a second letter, and 24 ways to choose a third letter.

\[
26 \times 25 \times 24
\]

Choosing \( r \) objects—some number of cards—from \( n \) objects—a full deck of cards or the alphabet in this case—is a truncated factorial. The easiest way to express this is like this:

\[n!\]As a reminder, \( n! \) is the notation for the factorial of \( n \). Factorials are computed by multiplying together all the integers up to \( n \). For example, \( 4! \) is calculated as follows: \( 4 \times 3 \times 2 \times 1 = 24 \).
(8.4) \[ \frac{n!}{(n - r)!} \]

The logic of this equation is that we do the full factorial in the numerator, and then divide out the part we are not interested in. In the first example at hand, we put 52! in the numerator, and then divide out 49! because we’re really only interested in \( 52 \times 51 \times 50 \). In the second example, we put 26! in the numerator and 23! in the denominator. This may seem like a whole lot of extra math, but it allows a general expression for these cases.

Notice that this distinguishes the order that we might draw the cards or select the letters. Thus the math above counts the same cards or letters in different orders as different possibilities. If we want to consider the possibility that we draw some number of cards or letters in any order, then we have to divide out the number of orders too. This means we have to divide out the number of permutations for \( r \) elements, which we’ve already seen is \( r! \). There is a special notation for this:

(8.5) \[ \binom{n}{r} = \frac{n!}{r!(n - r)!} \]

In the case of the three-card example, we have:

\[ \begin{align*} \binom{52}{3} &= \frac{52!}{3!(52 - 3)!} = \frac{52 \times 51 \times 50}{3 \times 2 \times 1} = \frac{132600}{6} = 22100 \end{align*} \]

In the case of the three-letter example, we have:

\[ \begin{align*} \binom{26}{3} &= \frac{26!}{3!(26 - 3)!} = \frac{26 \times 25 \times 24}{3 \times 2 \times 1} = \frac{15600}{6} = 2600 \end{align*} \]

### 8.4 Laws

We’ve already seen some of these, but let’s summarize the laws of probability that we’ve dealt with. First, if some event \( e \) exhibits probability \( p(e) \), then \( e \) not occurring, written \( p(\neg e) \), has probability \( 1 - p(e) \).
(8.6) \( p(\overline{e}) = 1 - p(e) \)

Thus, if the probability of throwing a six is \( \frac{1}{6} \), then the probability of not throwing a six is \( 1 - \frac{1}{6} = \frac{5}{6} \). If, for the sake of discussion, the probability of uttering a verb is .3, then the probability of uttering something that isn’t a verb is \( 1 - .3 = .7 \). If there are 6273 verbs out of 19528 words, then the probability of a verb is:

\[
\frac{6273}{19528} = .3212
\]

It follows that the probability of uttering something that is not a verb is:

\[
1 - .3212 = .6787
\]

If two events \( A \) and \( B \) are mutually exclusive (their outcomes do not overlap), then the probability that \( A \) or \( B \) occurs is:

(8.7) \( p(A \cup B) = p(A) + p(B) \)

Thus, if the probability of throwing a six is \( \frac{1}{6} \) and the probability of throwing a five is \( \frac{1}{6} \), then the probability of throwing a five or a six is \( \frac{1}{6} + \frac{1}{6} = \frac{2}{6} = \frac{1}{3} \). Likewise, if the probability of a word beginning with [p] is

\[
\frac{1704}{19528} = .0872
\]

and the probability of a word beginning with [t] is

\[
\frac{878}{19528} = .0449
\]

Then the probability of a word beginning with either [p] or [t] is \( .0872 + .0449 = .1321 \).

If two events \( A \) and \( B \) are independent, then the probability that \( A \) and \( B \) occurs is:
(8.8) \( p(A \cap B) = p(A) \times p(B) \)

This is also written \( p(A, B) \) and is referred to as the joint probability of \( A \) and \( B \). For example, if the probability of throwing a six is \( \frac{1}{6} \), then the probability of doing it twice in a row is \( \frac{1}{6} \times \frac{1}{6} = \frac{1}{36} \). If the probability that the first of two words begins with a \([p]\) is \( .0872 \) and the probability that the second word begins with a \([p]\) is the same, then the probability that both begin with a \([p]\) is:

\[
.0872 \times .0872 = .0076
\]

### 8.5 A Tricky Example

Here’s a tricky problem. Given \( n \) people in a room, what is the probability that at least two of those people will have the same birthday?

Let’s assume that there are four people in the room. The chances that they will have different birthdays are:

\[
\frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \frac{362}{365}
\]

Therefore the chances that at least two will have the same birthday (the chances of any other configuration) are:

\[
1 - \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \frac{362}{365} = 1 - \frac{365!}{(365 - 4)! \times 365^4} = 1 - .9834 = .0165
\]

The general form of this is that the probability of at least two people having the same birthday out of \( n \) people is:

\[
(8.9) \quad 1 - \frac{365!}{(365 - n)! \times 365^n}
\]
8.6 Conditional Probability

Let’s now consider the notion of conditional probability. The basic idea is that the probability of some event may be contingent on the probability of some other event. Consider, for example, the probability that some child should be a boy or girl. We can assume that the genders are equally likely. Given a family of two, where we know one child is a boy, what is the probability that the other child is a girl?

Consider the sample space; there are four equally probable situations.

(8.10) 1 girl, girl
2 girl, boy
3 boy, girl
4 boy, boy

Given that one child is a boy, we know we are in one of the last three situations. Among those, the other child is a girl in two out of three cases. Thus the probability that the other child is a girl is $\frac{2}{3} = 0.66$.

The conditional probability of $A$ given $B$ is written $p(A|B)$. The definition of conditional probability is the joint probability $p(A, B)$ divided by the prior probability $p(B)$.

(8.11) $p(A|B) = \frac{p(A, B)}{p(B)}$

We’ve already noted that the joint probability $p(A, B)$ can also be written $p(A \land B)$.

In the example at hand, we take $A$ to be the event where one child is a girl and $B$ as the event where one child is a boy. We have $p(A, B) = \frac{2}{4}$, the probability of the two kids being a boy and a girl. For the probability of a boy, we have $p(B) = \frac{3}{4}$. (This may seem odd, but follows from observing that in three out of the four possible situations, at least one child is a boy.) Thus:

(8.12) $p(A|B) = \frac{2}{\frac{4}{3}} = \frac{2}{4} \times \frac{3}{4} = \frac{8}{12} = \frac{2}{3} = 0.66$

As a language example, let’s consider the conditional probability that the second letter of a word is a vowel letter, given that the first letter is a
consonant letter. Let’s represent this as \( p(V_2|C_1) \). We use the text of the preceding paragraph (minus equations) for this. To do this, we must calculate \( p(C_1) \) and \( p(C_1, V_2) \).

There are 74 words in the paragraph, 45 of which begin with consonant letters. Hence the probability of an initial consonant letter is:

\[
p(C_1) = \frac{45}{74} = .608
\]

There are 26 words that begin with a consonant letter followed by a vowel letter. Hence the joint probability of an initial consonant letter and a following vowel letter is:

\[
p(C_1, V_2) = \frac{26}{74} = .351
\]

The conditional probability of a vowel letter in second position based on a consonant letter in first position is therefore:

\[
p(V_2|C_1) = \frac{p(C_1, V_2)}{p(C_1)} = \frac{.351}{.608} = .577
\]

Let’s now consider some interesting properties that follow from this definition. First, if events \( A \) and \( B \) are independent, then \( P(A|B) = p(A) \). This follows from the combination of independent events, given in (8.8) above. That is, if \( A \) and \( B \) are independent, then \( p(A, B) = p(A) \times p(B) \).

\[
(8.13) \quad p(A|B) = \frac{p(A, B)}{p(B)} = \frac{p(A) \times p(B)}{p(B)} = p(A)
\]

The first step above is just the definition of conditional probability. The second step replaces \( p(A, B) \) with \( p(A) \times p(B) \), since \( A \) and \( B \) are independent. In the last step, \( p(B) \) is cancelled from the numerator and denominator.

It also follows that the conditional probability of an event \( A \) given all possible events it could be conditional on is equivalent to \( p(A) \) directly. For example:

\[
(8.14) \quad p(A|B)p(B) + p(A|\overline{B})p(\overline{B}) = p(A)
\]
Here we treat $A$ in terms of some event $B$ occurring and the same event not occurring. This is, of course, equivalent to:

\begin{align}
(8.15) \quad p(A|B)p(B) + p(A|\overline{B})p(\overline{B}) &= \frac{p(A,B)}{p(B)}p(B) + \frac{p(A,\overline{B})}{p(B)}p(\overline{B}) \\
&= p(A, B) + p(A, \overline{B}) \\
&= p(A)
\end{align}

The first step follows from the definition of conditional probability. The second step cancels identical terms in the numerators and denominators. Finally the last step follows from the fact that $B$ and $\overline{B}$ exhaust the sample space.

Thomas Bayes (1702–1761) proposed this equivalence, which follows from what we have shown above:

\begin{align}
(8.16) \quad p(A|B) = \frac{p(B|A)p(A)}{p(B|A)p(A) + p(B|\overline{A})p(\overline{A})}
\end{align}

This is referred to as Bayes’ Law. Recall that the denominator above is equivalent to $p(B)$. Hence Bayes’ Law is equivalent to:

\[ p(A|B) = \frac{p(B|A)p(A)}{p(B)} \]

We then multiply both sides by $p(B)$ and get:

\[ p(A|B)p(B) = p(B|A)p(A) \]

Finally, from the definition of conditional probability, we know:

\[ p(A|B)p(B) = p(A, B) = p(B, A) = p(B|A)p(A) \]

Bayes’ Law is a very simple algebraic manipulation. The key conceptual point is that we can use it to compute $p(A|B)$ without know what $p(B)$ is.

Here is a simple example showing the utility of Bayes’ Law (Isaac, 1995).
Imagine we have a test for some disease. The test has a false positive rate of 5% (.05). This means that 5% of the people who get a positive response on the test actually do not have the disease. The distribution of the disease in the population is estimated at .3% (.003); 3 out of a thousand people in the general population actually have the disease. Assume as well that the true positive rate is .95: if we give the test to 100 people who have the disease, 95 of them will get a positive response on the test. If you test positive for the disease, what are the chances you actually have it?

Assume the following assignments:

\[
\begin{align*}
A &= \{\text{tested person has disease}\} \\
B &= \{\text{test result is positive}\} \\
p(A) &= .003 \\
p(\overline{A}) &= 1 - p(A) = .997 \\
p(B|\overline{A}) &= .05 \\
p(B|A) &= .95 \\
\end{align*}
\]

This gives us:

\[
p(A|B) = \frac{(.95)(.003)}{(.95)(.003) + (.05)(.997)} \approx .05
\]

The chance of a true positive, \(p(A|B)\) is only .05; if 100 people in the general population test positive for the disease, only 5 of them will actually have the disease. This is a rather surprising result. Given the low occurrence of the disease and the relatively high false positive rate, the chances of a true positive are fairly small.

### 8.7 Distributions

In this section, we define the notion of a probability distribution, the range of values that might occur in some space of outcomes. To do this, we need the notion of a random variable. Let’s define a random variable as follows.

**Definition 20 (Random variable)** A random variable is a function that assigns to each outcome in a sample space a unique number. Those numbers exhibit a probability distribution.
Consider an example. The sample space is the number of heads we might throw in three consecutive throws of a coin.

\[
\begin{array}{c|c}
\text{throw} & \text{number of heads} \\
(H, H, H) & 3 \\
(H, T, H) & 2 \\
(H, H, T) & 2 \\
(H, T, T) & 1 \\
(T, H, H) & 2 \\
(T, T, H) & 1 \\
(T, H, T) & 1 \\
(T, T, T) & 0 \\
\end{array}
\]

There is one way to get three heads, three ways to get two heads, three ways to get one heads, and one way to get no heads. The probability of throwing a head on one throw is \( p(H) = .5 \). The probability of throwing a tails is then \( p(T) = 1 - p(H) = .5 \). Call the random variable \( X \). The probability of throwing some number of heads can be computed as follows for each combination.

\[
p(X = 3) = 1 \cdot p(H)^3 \cdot p(T)^0 = 1 \cdot (.5)^3 \cdot (.5)^0 = 1 \cdot .125 \cdot 1 = .125 \\
p(X = 2) = 3 \cdot p(H)^2 \cdot p(T)^1 = 3 \cdot (.5)^2 \cdot (.5)^1 = 3 \cdot .25 \cdot .5 = .375 \\
p(X = 1) = 3 \cdot p(H)^1 \cdot p(T)^2 = 3 \cdot (.5)^1 \cdot (.5)^2 = 3 \cdot .5 \cdot .25 = .375 \\
p(X = 0) = 1 \cdot p(H)^0 \cdot p(T)^3 = 1 \cdot (.5)^0 \cdot (.5)^3 = 1 \cdot 1 \cdot .125 = .125
\]

The logic here is that you multiply together the number of possible combinations, the chances of success, raised to the same number, and the chances of failure, raised to the difference.

This is a binomial random variable. It describes the distribution of “successes” across some number of trials. If we have \( n \) trials and are interested in the likelihood of \( r \) successes, where the chance of success is \( p \) and the chance of failure is \( q = 1 - p \), then we have:

\[
p(X = r) = \frac{n!}{r!(n-r)!} \cdot p^r q^{n-r} = \binom{n}{r} \cdot p^r q^{n-r}
\]
Let’s try this with a linguistic application. Imagine we have a vocabulary of 100 words, 10 of which are verbs. All words are equally likely. All else being equal, in a sentence of four words, what are the chances that two of those words are verbs?

We assume \( n = 4, r = 2, \ p = .1, \ \text{and} \ q = .9 \). This gives us:

\[
(8.20) \quad p(X = 2) = \frac{4!}{2!(4-2)!} \cdot .1^2 \cdot .9^{4-2} \\
= \frac{24}{4} \cdot .01 \cdot .81 \\
= 6 \cdot .01 \cdot .81 \\
= .0486
\]

We plot the whole distribution for four trials below.

Let’s look at binomial distributions in graphical form. Here is the distribution for heads and tails for three throws.

\[
(8.21)
\]
(8.22)

Here it is for 100 throws.

(8.23)

And here it is for 1000 throws.
The binomial distribution is a discrete distribution. There are also continuous distributions, where there are an infinite number of points along the curve. The most useful one of these is the normal distribution.\footnote{The expression exp(n) is equivalent to $e^n$.}

\begin{equation}
(8.25) \quad f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)
\end{equation}

The variables to fill in here are $\mu$, the mean of the sample, and $\sigma$ the standard deviation of the sample.\footnote{The standard deviation is defined as: $\sqrt{\sum(x - \mu)^2}$, where $x$ is each value in the distribution.} Here is a picture of how this looks for a mean of 0 with points evenly distributed from $-5$ to 5. ($\mu = 0$ and $\sigma = 2.93$.)
The normal distribution shows up in all sorts of contexts and is very widely used in various statistical tests.

8.8 Summary

This chapter has introduced the basics of probability theory. We began with a general characterization of the notion and proceeded to a mathematical one.

We then considered some basic ideas of combinatorics, how to calculate the number of ways elements can be ordered or chosen.

We presented some basic laws of probability theory, including Bayes’ Law, and we defined the notions of joint probability and conditional probability.

Finally, we very briefly discussed the binomial and normal distributions.

8.9 Exercises

1. What is the probability of throwing a 3 with one throw of one die?

2. What is the probability of throwing a 3 and then another 3 with two throws of a single die?
3. What is the probability of throwing a 3 and then a 6 with two throws of a single die?

4. What is the probability of throwing anything but a 3 and another 3 with two throws of a single die?

5. What is the probability of not throwing a 3 at all in two throws of a single die?

6. What is the probability of throwing at least one 3 with two throws of a single die?

7. What is the probability of drawing a jack in one draw from a full deck of playing cards?

8. What is the probability of drawing four aces in four draws from a full deck of playing cards?

9. What is the probability of not drawing any aces in five draws from a full deck of playing cards?

10. How many people have to be in a room before the odds of at least two of them having the same birthday is greater than 60%?

11. For the following questions, assume this inventory:

   a  e  i  o  u
   p  t  k
   b  d  g
   m  n  ŋ

   (a) In a one-segment word, what are the odds of a word being [a]?

   (b) In a three-segment word, what are the odds of a word being [tap] vs. being [ppp]?

   (c) On this story, are segments more like dice or more like cards?

12. Identify some phenomenon in language, other than those exemplified in the chapter, that can be represented in terms of conditional probability.

13. Identify some phenomenon in language that can be seen as a random variable and give its empirical probability distribution (the distribution based on a sample of data).
Chapter 9

Probabilistic Language Models

In this chapter, we consider probability models that are specifically linguistic: *Hidden Markov Models* (HMMs) and *Probabilistic Context-free Grammars* (PCFGs).

These models can be used to directly encode probability values in linguistic formalism. While such models have well-understood formal properties and are widely used in computational research, they are extremely controversial in the theory of language. It is a hotly debated question whether grammar itself should incorporate probabilistic information.

9.1 The Chain Rule

To understand HMMs, we must first understand the *Chain Rule*. The Chain Rule is one simple consequence of the definition of conditional probability: the joint probability of some set of events $a_1, a_2, a_3, a_4$ can also be expressed as a ‘chain’ of conditional probabilities, e.g.:

\[
(p_1, p_2, p_3, p_4) = p(a_1)p(a_2|a_1)p(a_3|a_1, a_2)p(a_4|a_1, a_2, a_3)
\]

This follows algebraically from the definition of conditional probability. If we substitute by the definition of conditional probability for each of the conditional probabilities in the preceding equation and then cancel terms, we get the original joint probability.
(9.2) \[ p(a_1, a_2, a_3, a_4) = p(a_1) \times p(a_2 \mid a_1) \times p(a_3 \mid a_1, a_2) \times p(a_4 \mid a_1, a_2, a_3) \]
\[ = p(a_1) \times \frac{p(a_2 \mid a_1)}{p(a_1)} \times \frac{p(a_3 \mid a_1, a_2)}{p(a_1, a_2)} \times \frac{p(a_4 \mid a_1, a_2, a_3)}{p(a_1, a_2, a_3)} \]
\[ = p(a_1, a_2, a_3, a_4) \]

Notice also that the chain rule can be used to express any dependency among the terms of the original joint probability. For example:

(9.3) \[ p(a_1, a_2, a_3, a_4) = p(a_4)p(a_3 \mid a_4)p(a_2 \mid a_3, a_4)p(a_1 \mid a_2, a_3, a_4) \]

Here we express the first events as conditional on the later events, rather than vice versa. This array also follows algebraically:

(9.4) \[ p(a_1, a_2, a_3, a_4) = p(a_4) \times p(a_3 \mid a_4)p(a_2 \mid a_3, a_4)p(a_1 \mid a_2, a_3, a_4) \]
\[ = p(a_4) \times \frac{p(a_3 \mid a_4)}{p(a_4)} \times \frac{p(a_2 \mid a_3, a_4)}{p(a_3, a_4)} \times \frac{p(a_1 \mid a_2, a_3, a_4)}{p(a_2, a_3, a_4)} \]
\[ = p(a_1, a_2, a_3, a_4) \]

9.2 N-gram models

Another preliminary to both HMMs and PCGFs are N-gram models. These are the very simplest kind of statistical language model. The basic idea is to consider the structure of a text, corpus, or language as the probability of different words occurring alone or in sequence. The simplest model, the unigram model, treats words in isolation.

9.2.1 Unigrams

Take a very simple text like the following:

Peter Piper picked a peck of pickled pepper.
Where’s the pickled pepper that Peter Piper picked?

There are sixteen words in this text.\(^1\) The word “Peter” occurs twice and thus has a probability of \( \frac{2}{16} = .125 \). On the other hand, “peck” occurs only once and has the probability of \( \frac{1}{16} = .0625 \).

---

\(^1\)We leave aside issues of text normalization, i.e. capitalization and punctuation.
This kind of information can be used to judge the well-formedness of texts. The way this works is that we calculate the overall probability of the new text as a function of the individual probabilities of the words that occur in it.

On this view, the likelihood of a text like “Peter pickled pepper” would be a function of the probabilities of its parts: .125, .125, and .125. If we assume the choice of each word is independent, then the probability of the whole string is the product of the independent words, in this case: .125 × .125 × .125 = .00195.

We can make this more intuitive by considering a restricted hypothetical case. Imagine we have a language with only three words: \{a, b, c\}. Each word has an equal likelihood of occurring: .33 each. There are nine possible two-word texts, each having a probability of .1089. The range of possible texts thus exhibits a probability distribution; the probabilities of the set of possible outcomes sums to 1 (9 × .1089).

\[
\begin{array}{cc}
 xy & p(x) \times p(y) \\
 aa & .1089 \\
 ab & .1089 \\
 ac & .1089 \\
 ba & .1089 \\
 bb & .1089 \\
 bc & .1089 \\
 ca & .1089 \\
 cb & .1089 \\
 cc & .1089 \\
 total & 1 \\
\end{array}
\]

This is also the case if the individual words exhibit an asymmetric distribution. Assume a vocabulary with the same words, but where the individual word probabilities are different, i.e. \(p(a) = .5\), \(p(b) = .25\), and \(p(c) = .25\).
The way this model works is that the well-formedness of a text fragment is correlated with its overall probability. Higher-probability text fragments are more well-formed than lower-probability texts, e.g. $aa$ is a better exemplar of $L$ than $cb$.

A major shortcoming of this model is that it makes no distinction among texts in terms of ordering. Thus this model cannot distinguish $ab$ from $ba$. This, of course, is something that we as linguists think is essential, but we can ask the question whether it’s really necessary for computational applications.

A second major shortcoming is that the model makes the same predictions at any point in the text. For example, in the second example above, the most frequent word is $a$. Imagine we want to predict the first word of some text. The model above would tell us it should be $a$. Imagine we want to predict the $n$th word. Once again, the model predicts $a$. The upshot is that the current model predicts that a text should simply be composed of the most frequent item in the lexicon: $aaa . . .$.

### 9.2.2 Bigrams

Let’s go on to consider a more complex model that captures some of the ordering restrictions that may occur in some language or text: bigrams. The basic idea behind higher-order N-gram models is to consider the probability of a word occurring as a function of its immediate context. In a bigram model, this context is the immediately preceding word:

$$
\begin{array}{|c|c|}
\hline
xy & p(x) \times p(y) \\
\hline
aa & .25 \\
ab & .125 \\
ac & .125 \\
ba & .125 \\
bb & .0625 \\
bc & .0625 \\
ca & .125 \\
\hline
\end{array}
$$

Note that we have made an interesting leap here. We are characterizing “well-formedness” in terms of frequency. Is this fair?
CHAPTER 9. PROBABILISTIC LANGUAGE MODELS

\[(9.7)\] 
\[p(w_1w_2 \ldots w_i) = p(w_1) \times p(w_2|w_1) \times \ldots \times p(w_i|w_{i-1})\]

We calculate conditional probability in the usual fashion. (We use absolute value notation to denote the number of instances of some element.)

\[(9.8)\] 
\[p(w_i|w_{i-1}) = \frac{p(w_{i-1}, w_i)}{p(w_{i-1})} = \frac{|w_{i-1}w_i|}{|w_{i-1}|}\]

It’s important to notice that this is not the Chain Rule; here the context for the conditional probabilities is the immediate context, not the whole context. As a consequence we cannot return to the joint probability algebraically, as we did at the end of the preceding section (equation 9.2 on page 168). We will see that this limit on the context for the conditional probabilities in a higher-order N-gram model has important consequences for how we might actually manipulate such models computationally.

Let’s work out the bigrams in our tongue twister (repeated here).

Peter Piper picked a peck of pickled pepper.
Where’s the pickled pepper that Peter Piper picked?

The frequencies of individual words are given in the table below:

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>2</td>
</tr>
<tr>
<td>Piper</td>
<td>2</td>
</tr>
<tr>
<td>picked</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>peck</td>
<td>1</td>
</tr>
<tr>
<td>of</td>
<td>1</td>
</tr>
<tr>
<td>pickled</td>
<td>2</td>
</tr>
<tr>
<td>pepper</td>
<td>2</td>
</tr>
<tr>
<td>Where’s</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>that</td>
<td>1</td>
</tr>
</tbody>
</table>

The bigram frequencies are:
### Bigrams and Bigram Frequencies

<table>
<thead>
<tr>
<th>Bigrams</th>
<th>Bigram Frequencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>picked a</td>
<td>1</td>
</tr>
<tr>
<td>pepper that</td>
<td>1</td>
</tr>
<tr>
<td>peck of</td>
<td>1</td>
</tr>
<tr>
<td>a peck</td>
<td>1</td>
</tr>
<tr>
<td>pickled pepper</td>
<td>2</td>
</tr>
<tr>
<td>Where s</td>
<td>1</td>
</tr>
<tr>
<td>Piper picked</td>
<td>2</td>
</tr>
<tr>
<td>the pickled</td>
<td>1</td>
</tr>
<tr>
<td>Peter Piper</td>
<td>2</td>
</tr>
<tr>
<td>of pickled</td>
<td>1</td>
</tr>
<tr>
<td>pepper Where</td>
<td>1</td>
</tr>
<tr>
<td>that Peter</td>
<td>1</td>
</tr>
<tr>
<td>s the</td>
<td>1</td>
</tr>
</tbody>
</table>

Calculating conditional probabilities is then a straightforward matter of division. For example, the conditional probability of “Piper” given “Peter”:

\[
p(Piper|Peter) = \frac{|Peter Piper|}{|Peter|} = \frac{2}{2} = 1
\]

However, the conditional probability of “Piper” given “a”:

\[
p(Piper|a) = \frac{|a Piper|}{|a|} = 0 \quad \frac{1}{1} = 0
\]

Using conditional probabilities thus captures the fact that the likelihood of “Piper” varies by preceding context: it is more likely after “Peter” than after “a”.

The bigram model addresses both of the problems we identified with the unigram model above. First, recall that the unigram model could not distinguish among different orderings. Different orderings are distinguished in the bigram model. Consider, for example, the difference between “Peter Piper” and “Piper Peter”. We’ve already seen that the former has a conditional probability of 1. The latter, on the other hand:

\[
p(Peter|Piper) = \frac{|Piper Peter|}{|Piper|} = \frac{0}{2} = 0
\]
The unigram model also made the prediction that the most well-formed text should be composed of repetitions of the highest-probability items. The bigram model excludes this possibility as well. Contrast the conditional probability of “Peter Piper” with “Peter Peter”:

\[
p(Peter|Peter) = \frac{|Peter\ Peter|}{|Peter|} = \frac{0}{2} = 0
\]

The bigram model presented doesn’t actually give a probability distribution for a string or sentence without adding something for the edges of sentences. To get a correct probability distribution for the set of possible sentences generated from some text, we must factor in the probability that some word begins the sentence, and that some word ends the sentence. To do this, we define two markers that delimit all sentences: \(<s>\) and \(</s>\). This transforms our text as follows.

\(<s>\) Peter Piper picked a peck of pickled pepper. \(</s>\)
\(<s>\) Where’s the pickled pepper that Peter Piper picked? \(</s>\)

Thus the probability of some sentence \(w_1w_2\ldots w_n\) is given as:

\[
p(w_1|<s>) \times p(w_2|w_1) \times \ldots \times p(</s>|w_n)
\]

Given the very restricted size of this text, and the conditional probabilities dependent on the newly added edge markers, there are only several specific ways to add acceptable strings without reducing the probabilities to zero. Given the training text, these have the probabilities given below:

\[
p(\text{Peter Piper picked}) = p(\text{Peter}|<s>) \times p(\text{Piper}|\text{Peter}) \times p(\text{picked}|\text{Piper}) \times p(</s>|\text{picked})
= .5 \times 1 \times 1 \times .5
= .25
\]
(9.17) \[ p(\text{Where's the pickled pepper}) \]
\[ = p(\text{Where's}|<s>) \times p(\text{the}|\text{Where's}) \times p(\text{pickled}|\text{the}) \times p(\text{pepper}|\text{pickled}) \times p(<s>|\text{pepper}) \]
\[ = .5 \times 1 \times 1 \times 1 \times .5 \]
\[ = .25 \]

(9.18) \[ p \left( \frac{\text{Where's the pickled pepper that Peter}}{\text{Piper picked a peck of pickled pepper}} \right) \]
\[ = p(\text{Where's}|<s>) \times p(\text{the}|\text{Where's}) \times p(\text{pickled}|\text{the}) \times p(\text{pepper}|\text{pickled}) \times p(\text{that}|\text{pepper}) \times p(\text{Peter}|\text{that}) \times p(\text{Piper}|\text{Peter}) \times p(\text{picked}|\text{Piper}) \times p(\text{a}|\text{picked}) \times p(\text{peck}|\text{a}) \times p(\text{of}|\text{peck}) \times p(\text{pickled}|\text{of}) \times p(\text{pepper}|\text{pickled}) \times p(<s>|\text{pepper}) \]
\[ = .5 \times 1 \times 1 \times .5 \times 1 \times 1 \times .5 \times 1 \times 1 \times 1 \times .5 \]
\[ = .0625 \]

(9.19) \[ p \left( \frac{\text{Peter Piper picked a peck of pickled pepper}}{\text{that Peter Piper picked}} \right) \]
\[ = p(\text{Peter}|<s>) \times p(\text{Piper}|\text{Peter}) \times p(\text{picked}|\text{Piper}) \times p(\text{a}|\text{picked}) \times p(\text{peck}|\text{a}) \times p(\text{of}|\text{peck}) \times p(\text{pickled}|\text{of}) \times p(\text{pepper}|\text{pickled}) \times p(\text{that}|\text{pepper}) \times p(\text{Piper}|\text{Peter}) \times p(\text{picked}|\text{Piper}) \times p(<s>|\text{picked}) \]
\[ = .5 \times 1 \times 1 \times .5 \times 1 \times 1 \times 1 \times .5 \times 1 \times 1 \times .5 \]
\[ = .0625 \]

Notice how the text that our bigram frequencies were calculated on only leaves very restricted “choice” points. There are really only four:

1. What is the first word of the sentence: “Peter” or “Where’s”?
2. What is the last word of the sentence: “pepper” or “picked”?  
3. What follows the word “picked”: “a” or “/>”?  
4. What follows the word “pepper”: “that” or “/>”?  
The only sequences of words allowed are those sequences that occur in the training text. However, even with these restrictions, this allows for sentences of unbounded length; an example like 9.19 can be extended infinitely.  
Notice, however, that these restrictions do not guarantee that all sentences produced in conformity with this language model will be grammatical (by normal standards); example (9.18) is ungrammatical. Since the only dependencies captured in a bigram model are local/immediate ones, such sentences will emerge as well-formed.\(^3\)

### 9.2.3 Higher-order N-grams

N-gram models are not restricted to unigrams and bigrams; higher-order N-gram models are also used. These higher-order models are characterized as we would expect. For example, a trigram model would view a text \(w_1w_2\ldots w_n\) as the product of a series of conditional probabilities:

\[
p(w_1w_2\ldots w_n) = p(w_1) \times p(w_2|w_1) \times \prod p(w_n|w_{n-2}w_{n-1})
\]

The advantage of higher-order N-gram models is that relationships spanning longer sequences of words can be captured. For example, a trigram model allows us to capture relationships between spans of three words long.  
The disadvantage of higher-order models is that much much larger training texts are required and it is virtually impossible to avoid accidental gaps.

### 9.2.4 N-gram approximation

One way to try to appreciate the success of N-gram language models is to use them to approximate text in a generative fashion. That is, we can compute all the occurring N-grams over some text, and then use those N-grams to generate new text.\(^4\)

\(^3\)Notice too, of course, that very simple sentences compatible with the vocabulary of this text would receive a null probability, e.g. “Where’s the pepper?” or “Peter picked a peck”, etc.  
\(^4\)See Shannon (1951).
Here is an example. The following ten “sentences” were constructed in the following way. The frequencies of all words in the short story “White Fang” by Jack London were calculated. Then each sentence was constructed by taking a random sample of those words whose frequency was above .002.

1. so her they dog no but there with in so
2. as not him they so he a that away then
3. be when dogs then up there he fang by a
4. on dogs out his and out he the away out
5. they then that on his into upon been their she
6. fang him this up dogs were he dogs no
7. by fang to into when him their when upon
8. up them at the was a been with there down
9. then down be him and on time one as into
10. as them be to for were that his at when

It’s easy to see that, though the words used are relatively common, these are hardly compelling as plausible sentences of English.

The following ten examples were constructed by a similar technique using bigrams extracted from the same text. Here the bigram frequencies exceed .0002.

1. half feet on everywhere upon itself as strongest dog
2. far outweighed a hostile movement beside scott you know
3. judge unknown was because it toward personal life
4. everybody gave himself to cheapen himself off with
5. it bristled fiercely belligerent and save once and death
6. because they spoke matt should be used his tail
7. turn ’m time i counted the horse up live
8. beast that cautiously it discovered an act of plenty
9. fatty’s gone before had thought in matt argued stubbornly
10. what evil that night was flying brands from weakness

Notice that this latter set of sentences is far more natural sounding.

Finally, here are ten examples constructed from trigrams taken from the same text. Here, trigram frequencies exceed .00002.

1. you was a thing to be the sign of
2. yet white fang with a wolf that knows enough
3. year when the master white fang he was not
4. void in his teeth to the ground white fang
5. upon the pack but he was no way of
6. two of them was the will of the bush
7. suspicious of them and the cub could not understand
8. rushed upon him with the dog with which to
9. put him into the world it was different from
10. overture to white fang and the dogs and to

These are even more natural-sounding.

### 9.3 Hidden Markov Models

In this section, we treat *Hidden Markov Models* (HMMs). These are intimately associated with N-gram models and widely used as a computational model for language processing.
9.3.1 Markov Chains

To understand Hidden Markov Models, we must first understand Markov chains. These are basically DFAs with associated probabilities. Each arc is associated with a probability value and all arcs leaving any particular node must exhibit a probability distribution, i.e. their values must range between 0 and 1, and must total 1. In addition, one node is designated as the “starting” node. There is no set of designated final states. A simple example is given below.

As with an FSA, the machine moves from state to state following the arcs given. The sequence of arc symbols denotes the string generated—accepted—by the machine. The difference between a Markov chain and an FSA is that in the former case there are probabilities associated with each arc. These probabilities are multiplied together to produce the probability that the machine might follow any particular sequence of arcs/states, generating the appropriate string. For example, the probability of producing a single $a$ and returning to $s_1$ is .3; the probability of going from $s_1$ to $s_2$ and emitting a $b$ is .7. Hence the probability of $ab$ is $0.3 \times 0.7 = 0.21$. The probability of producing the sequence $ba$, however, is $0.7 \times 0.2 = 0.14$. In the first case we go from $s_1$ to $s_1$ to $s_2$; in the second case from $s_1$ to $s_2$ to $s_1$.

There are several key facts to note about a Markov chain. First, as we stated above, the probabilities associated with the arcs from any state exhibit a probability distribution. For example, in the Markov chain above, the arcs from $s_1$ are $0.3 + 0.7 = 1$. Second, Markov chains are analogous to a deterministic finite state automaton: there are no choices at any point either in terms of start state or in terms of what arcs to follow. Thus there is precisely one and only one arc labeled with each symbol in the alphabet from each state in the chain. Third, it follows that any symbol string uniquely determines a state sequence. That is, for any particular string, there is one and only one corresponding sequence of states through the chain.
9.3.2 Hidden Markov Models

It’s also possible to imagine a non-deterministic Markov chain; these are referred to as Hidden Markov Models (HMMs). Once we’ve introduced indeterminacy anywhere in the model, we can’t uniquely identify a state sequence for all strings. A legal string may be compatible with several paths; hence the state sequence is “hidden”. Given the model above, we can introduce indeterminacy in several ways. First, we can allow for multiple start states.

The example below is of this sort. Here each state is associated with a “start” probability. (Those must, of course, exhibit a probability distribution and sum to 1.) This means, that for any particular string, one must factor in all possible start probabilities. For example, a string $b$ could be generated/accepted by starting in $s_1$ and then following the arc to $s_2$ (.4 × .7 = .28). We could also start in $s_2$ and then follow the arc back to $s_1$ (.6 × .8 = .48).

\[ \text{(9.22)} \]

The overall probability of the string $b$ is the sum of the probabilities of all possible paths through the HMM: .28 + .48 = .76. Notice then that we cannot really be sure which path may have been taken to get to $b$, though if the paths have different probabilities then we can calculate the most likely path. In the case at hand, this is $s_2 \vdash s_1$.

Indeterminacy can also be introduced by adding multiple arcs from the same state for the same symbol. For example, the HMM below is a minimal modification of the Markov chain above.

\[ \text{(9.23)} \]

Consider how this HMM deals with a string $ab$. Here only $s_1$ is a legal start state. We can generate/accept $a$ by either following the arc back to $s_1$ (.3) or by following the arc to $s_2$ (.2). In the former case, we can get $b$ by following the arc from $s_1$ to $s_2$. In the latter case, we can get $b$ by following the arc
from \( s_2 \) back to \( s_1 \). This gives the following total probabilities for the two state sequences given.

\[
\begin{align*}
(9.24) \quad s_1 \rightarrow s_1 \rightarrow s_2 &= .3 \times .5 = .15 \\
&= s_1 \rightarrow s_2 \rightarrow s_1 = .2 \times .8 = .16
\end{align*}
\]

This results in an overall probability of .31 for \( ab \). The second state sequence is of course the more likely one since it exhibits a (slightly) higher overall probability.

A HMM can naturally include both extra arcs and multiple start states. The HMM below exemplifies.

\[
\begin{align*}
(9.25)
\end{align*}
\]

This generally results in even more choices for any particular string. For example, the string \( ab \) can be produced with all the following sequences:

\[
\begin{align*}
(9.26) \quad s_1 \rightarrow s_1 \rightarrow s_2 &= .1 \times .3 \times .5 = .015 \\
&= s_1 \rightarrow s_2 \rightarrow s_1 = .1 \times .2 \times .8 = .016 \\
&= s_2 \rightarrow s_1 \rightarrow s_2 = .9 \times .2 \times .5 = .09 \\
\end{align*}
\]

The overall probability is then .121.

### 9.3.3 Formal HMM properties

There are a number of formal properties of Markov chains and HMMs that are useful. One extremely important property is *Limited Horizon*:

\[
(9.27) \quad p(X_{t+1} = s_k | X_1, \ldots, X_t) = p(X_{t+1} = s_k | X_t)
\]

This says that the probability of some state \( s_k \) given the set of states that have occurred before it is the same as the probability of that state given the single state that occurs just before it.
As a consequence of the structure of a HMM, there is a probability distribution over strings of any particular length:

\[ \forall n \sum_{w_{1:n}} p(w_{1:n}) = 1 \]

What this means is that when we sum the probabilities of all possible strings of any length \( n \), their total is 1.

For example, consider the set of strings two characters long with respect to the HMM in (9.25).

<table>
<thead>
<tr>
<th>string</th>
<th>path</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>( aa )</td>
<td>( s_1 \vdash s_1 \vdash s_1 )</td>
<td>.1 \times .3 \times .3 = .009</td>
</tr>
<tr>
<td></td>
<td>( s_1 \vdash s_1 \vdash s_2 )</td>
<td>.1 \times .3 \times .2 = .006</td>
</tr>
<tr>
<td></td>
<td>( s_1 \vdash s_2 \vdash s_1 )</td>
<td>.1 \times .2 \times .2 = .004</td>
</tr>
<tr>
<td></td>
<td>( s_2 \vdash s_1 \vdash s_1 )</td>
<td>.9 \times .2 \times .3 = .054</td>
</tr>
<tr>
<td></td>
<td>( s_2 \vdash s_1 \vdash s_2 )</td>
<td>.9 \times .2 \times .2 = .036</td>
</tr>
<tr>
<td>( ab )</td>
<td>( s_1 \vdash s_1 \vdash s_2 )</td>
<td>.1 \times .3 \times .5 = .015</td>
</tr>
<tr>
<td></td>
<td>( s_1 \vdash s_2 \vdash s_1 )</td>
<td>.1 \times .2 \times .8 = .016</td>
</tr>
<tr>
<td></td>
<td>( s_2 \vdash s_1 \vdash s_s )</td>
<td>.9 \times .2 \times .5 = .09</td>
</tr>
<tr>
<td>( ba )</td>
<td>( s_1 \vdash s_2 \vdash s_1 )</td>
<td>.1 \times .5 \times .2 = .01</td>
</tr>
<tr>
<td></td>
<td>( s_2 \vdash s_1 \vdash s_1 )</td>
<td>.9 \times .8 \times .3 = .216</td>
</tr>
<tr>
<td></td>
<td>( s_2 \vdash s_1 \vdash s_2 )</td>
<td>.9 \times .8 \times .2 = .144</td>
</tr>
<tr>
<td>( bb )</td>
<td>( s_1 \vdash s_2 \vdash s_1 )</td>
<td>.1 \times .5 \times .8 = .04</td>
</tr>
<tr>
<td></td>
<td>( s_1 \vdash s_2 \vdash s_1 )</td>
<td>.9 \times .8 \times .5 = .36</td>
</tr>
</tbody>
</table>

9.3.4 Bigrams and HMMs

It is a straightforward matter to treat bigram models in terms of HMMs. In fact, we can simplify our model considerably and still get the right effect. Let us assume that the name of each state corresponds to a symbol in the alphabet. All arcs leading to some state \( s \) would thus be labeled \( s \); for convenience, we leave this label off.
Imagine now that we have a vocabulary of three words \( \{a, b, c\} \). We simply create a HMM with a state for each item in the vocabulary and then arcs indicate the conditional probability of each bigram. Thus an arc from state \( s_i \) to state \( s_j \) indicates the conditional probability: \( p(s_j|s_i) \). An example is given below. Here, for example, the conditional probability \( p(b|a) = .5 \). A complete text given this model would get an overall probability in the usual fashion.

9.3.5 Higher-order N-grams

How would such a model be extended to higher-order N-grams? At first blush, we might think there’s a problem. After all, the limited horizon property says that the history an HMM is sensitive to can be restricted to the immediately preceding state. A trigram model would appear to require more.

This, however, doesn’t reckon with the assumption of a finite vocabulary (albeit a large finite vocabulary). In the previous example, we took each state as equivalent to a vocabulary item. To treat a trigram model, we must allow for states to be equivalent to both single words in the vocabulary and every possible combination of words in the vocabulary. For example, to construct a HMM for a trigram model for the same vocabulary as the previous examples, we would augment the model to include nine additional states representing each combination of words. This is shown below. (Probabilities have been left off to enhance legibility.)
Context-free grammars can also be converted into statistical models: probabilistic context-free grammars (PCFGs). In this section we consider the structure of these models.

A PCFG is a context-free grammar where each rule has an associated probability. In addition, the rules that expand any particular non-terminal \( A \) must exhibit a probability distribution, i.e. their probabilities must sum to one (Suppes, 1970).

Let’s exemplify this with a very simple grammar of a subset of English. This grammar produces transitive and intransitive sentences with two verbs and two proper nouns.
(9.32) \[ S \rightarrow NP \ VP \]
\[ VP \rightarrow V \]
\[ VP \rightarrow V \ NP \]
\[ V \rightarrow \text{sees} \]
\[ V \rightarrow \text{helps} \]
\[ NP \rightarrow \text{Mindy} \]
\[ NP \rightarrow \text{Mary} \]

This produces parse trees as follows for sentences like Mindy sees Mary.

(9.33) \[
\begin{array}{c}
S \\
\downarrow \\
NP \\
\downarrow \\
\text{Mindy} \\
\downarrow \\
\text{sees} \\
\downarrow \\
\text{Mary} \\
\end{array}
\]

\[ VP \rightarrow \text{sees} \]
\[ VP \rightarrow \text{helps} \]
\[ NP \rightarrow \text{Mindy} \]
\[ NP \rightarrow \text{Mary} \]

To convert this into a probabilistic context-free grammar, we simply associate each production rule with a probability, such that—as noted above—the probabilities for all rules expanding any particular non-terminal sum to one. A sample PCFG that satisfies these properties is given below. Notice how the single rule expanding \( S \) has a probability of 1, since there is only one such rule. In all the other cases, there are two rules expanding each non-terminal and the probabilities associated with each pair sum to 1.

(9.34) \[ S \rightarrow NP \ VP \quad 1 \]
\[ VP \rightarrow V \quad .3 \]
\[ VP \rightarrow V \ NP \quad .7 \]
\[ V \rightarrow \text{sees} \quad .4 \]
\[ V \rightarrow \text{helps} \quad .6 \]
\[ NP \rightarrow \text{Mindy} \quad .2 \]
\[ NP \rightarrow \text{Mary} \quad .8 \]
The probability of some parse of a sentence is then the product of the probabilities of all the rules used. In the example at hand, *Mindy sees Mary*, we get: $1 \times .2 \times .7 \times .4 \times .8 = .0448$. The probability of a sentence $s$, e.g. any particular string of words, is the sum of the probabilities of all its parses $t_1, t_2, \ldots, t_n$.

\begin{equation}
(9.35) \quad p(s) = \sum_j p(t_j)p(s|t_j)
\end{equation}

In the case at hand, there are no structural ambiguities; there is only one possible structure for any acceptable sentence. Let’s consider another example, but one where there are structural ambiguities. The very simplified context-free grammar for noun conjunction below has these properties.

\begin{align*}
(9.36) \quad NP & \rightarrow NP \ C \ NP \ \ .4 \\
NP & \rightarrow Mary \ \ .3 \\
NP & \rightarrow Mindy \ \ .2 \\
NP & \rightarrow Mark \ \ .1 \\
C & \rightarrow and \ \ 1
\end{align*}

This grammar results in multiple trees for conjoined nouns like *Mary and Mindy and Mark* as below. The ambiguity surrounds whether the first two conjuncts are grouped together or the last two. The same rules are used in each parse, so the probability of either one of them is: $.3 \times .2 \times .1 \times 1 \times 1 \times .4 \times .4 = .00096$. The overall probability of the string is then $.00096 + .00096 = .00192$.

\textsuperscript{5}Note that if any rule is used more than once then it’s probability is factored in as many times as it is used.
Notice that the probability values get problematic when the PCFG is recursive, that is, when the grammar generates an infinite number of sentences. It then follows that at least some parses have infinitely small values. Let’s consider a toy grammar that allows recursive clausal embedding. This grammar allows optional recursion on $S$, but a very restricted vocabulary.

Do we get a probability distribution? Let’s look at a few examples:
We have $p(\text{John knows}) = 1 \times 1 \times 1 \times 0.6 \times 1 = 0.6$

We have $p(\text{John knows John knows}) = 1 \times 1 \times 0.4 \times 1 \times 1 \times 1 \times 0.6 \times 1 = 0.24$. 
We have $p(\text{John knows John knows John knows}) = 1 \times 1 \times 1 \times .4 \times 1 \times 1 \times 1 \times 1 \times 1 \times .6 = .096$.

You can see that every time we add a new clause, the probability value of the new sentence is the previous sentence multiplied by .4. This gives us this chart showing the decrease in probabilities as a function of the number of clauses.

<table>
<thead>
<tr>
<th>(9.43) clauses</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.6</td>
</tr>
<tr>
<td>2</td>
<td>.24</td>
</tr>
<tr>
<td>3</td>
<td>.096</td>
</tr>
<tr>
<td>4</td>
<td>.0384</td>
</tr>
<tr>
<td>5</td>
<td>.0153</td>
</tr>
<tr>
<td>6</td>
<td>.0061</td>
</tr>
<tr>
<td>$n$</td>
<td>?</td>
</tr>
</tbody>
</table>

We need some calculus to prove whether, in the limit, this totals to 1.
Chi (1999) shows formally that PCFGs don’t exhibit a probability distribution. He argues as follows. First, assume a grammar with only two rules:

\begin{align}
S & \rightarrow S \ S \\
S & \rightarrow a
\end{align}

If \( p(S \rightarrow S \ S) = n \) then \( p(S \rightarrow a) = 1 - n \). Chi argues as follows: “Let \( x_h \) be the total probability of all parses with height no larger than \( h \). Clearly, \( x_h \) is increasing. It is not hard to see that \( x_{h+1} = (1 - n) + nx_h^2 \). Therefore, the limit of \( x_h \), which is the total probability of all parses, is a solution for the equation \( x = (1 - n) + nx^2 \). The equation has two solutions: 1 and \( \frac{1}{n} - 1 \). It can be shown that \( x \) is the smaller of the two: \( x = \min(1, \frac{1}{n} - 1) \). Therefore, if \( n > \frac{1}{2} \), \( x < 1 \)—an improper probability.”

9.5 Summary

This chapter links the ideas about probability from the preceding chapter with the notions of grammar and automaton developed earlier.

We began with the notion of N-gram modeling. The basic idea is that we can view a strings of words or symbols in terms of the likelihood that each word might follow the preceding one or two words. While this is a staggeringly simple idea, it actually can go quite some distance toward describing natural language data.

We next turned to Markov Chains and Hidden Markov Models (HMMs). These are probabilistically weighted finite state devices and can be used to associate probabilities with FSAs.

We showed in particular how a simplified Markov Model could be used to implement N-grams, showing that those models could be treated in restrictive terms.

Finally, we very briefly considered probabilistic context-free grammars.

9.6 Exercises

1. Give a hypothetical example showing how the limited horizon property violates the chain rule, how it does not give a probability distribution.
2. Construct a unigram model for a very limited text.

3. The N-gram approximation examples above did not treat the edges of sentences differently from the interior of sentences. What effect do you expect from this?

4. N-gram models fail to capture certain kinds of generalizations about language. Can you cite an example?

5. Theories of sentence structure are sometimes complicated to include movement, the possibility that words or phrases can change location in a parse tree. If you are familiar with such theories, how might they be expressed probabilistically?
Bibliography


