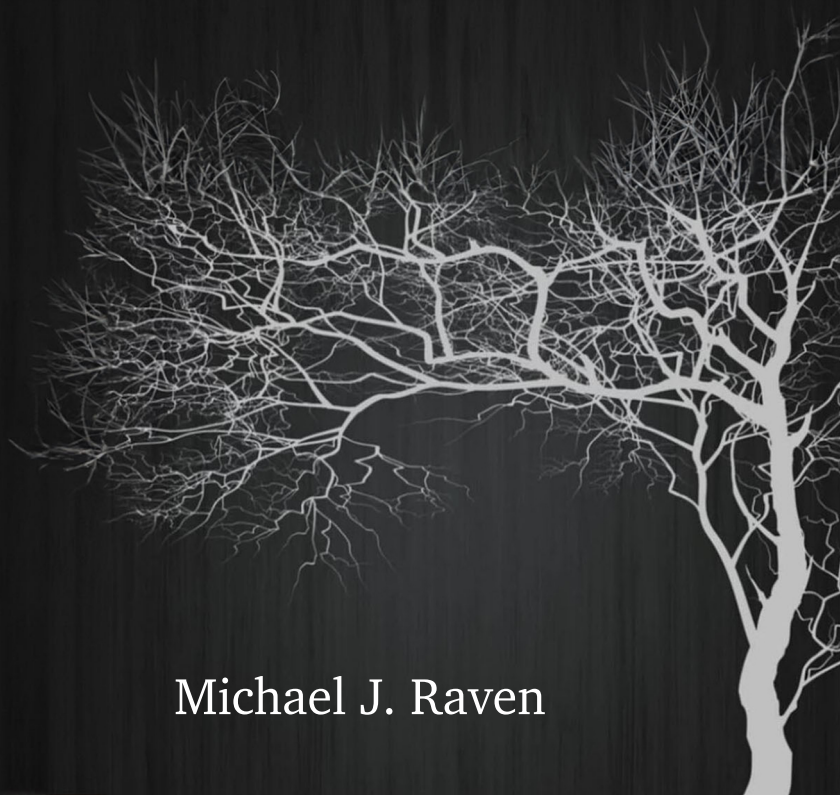
form

introducing formal logic



Michael J. Raven

form

Introducing Formal Logic

Michael J. Raven

2024

This version is revision ca72500 (2024-03-15;mjraven).

Cover design by Rose Choi.

Contents

Preface

This textbook introduces formal logic (specifically, sentential logic, monadic logic, and first-order logic). It was originally loosely based on forallx: *Calgary*, but has since diverged significantly from it. What's most distinctive about this book is its compartmentalizing the theory of logical form from its applications.

Acknowledgments Thanks to Mark Hinchliff, Paul Hovda, Conor Mayo-Wilson, and Cliff Roberts for helpful discussions about this book. Thanks also to Hartry Field, Kit Fine, and Crispin Wright who, in predictably different ways, influenced my conception of what logic is and how to teach it. And thanks to Rose Choi for the beautiful cover art.

For Instructors The syntax, semantics, and derivational systems are supported by Graham Leach-Krouse's free, online logic teaching software application *Carnap* (carnap.io). This allows for submission and automated marking of exercises such as semantic tables, derivations, and renderings.

PART I

Introducing Logical Form

CHAPTER 1
Logic

Many logic textbooks begin with grand pronouncements like:

Logic is about good reasoning.
Rational inquiry depends on logic.
Logic is the science of rational argumentation.

None of these is correct.

They all contain kernels of truth. Logic is *somehow* connected to reasoning, rationality, and argumentation. But, as the philosopher Gilbert Harman often warned, the pronouncements exaggerate and distort the connections. But then what is logic?

Logic is the study of **form**, or at least form of a certain sort. Form has intrigued scholars for milennia. It boasts deep applications to reasoning, rationality, and argumentation. These are so profound that an entire branch of philosophy, *logic*, is devoted to form. Why, then, do logic textbooks so often begin by distorting their subject?

It may be yet another case of confusing theory with application. Logic is abstract. Its relevance may be doubted until its applications are demonstrated. Nowadays, its most famous application is the digital computer. The digital age emerged from logic. Although emergence was pioneered by philosophers and logicians, its rapid successes spawned off new fields, like computer science and engineering. It is philosophy's fate to cede territory to new disciplines. One unceded (but disputed) territory is logic's application to reasoning. Even folks impatient with abstract studies may still be interested in reasoning. An eagerness to market logic to the impatient may suggest why the pronouncements are made. The connections between logic and reasoning are intricate and controversial. Simplifying them may streamline the marketing, even if it sows seeds that grow to distort the subject.

A deeper explanation comes from a potted history of logic. It begins with Aristotle, who was first to study form and its applica-

tions. The Stoics also made some major contributions. But the most dramatic developments began in the late 1800's with Gottlob Frege's pioneering work. His life project was to secure the foundations of mathematics. His strategy, *logicism*, was to show that mathematics was a branch of logic. To do this, he developed logic as we now know it. He then applied it to mathematics: he gave purely logical definitions of key mathematical notions and used logic alone to prove its axioms and theorems. In one way, Frege's project failed. Bertrand Russell's discovery of what is now known as *Russell's paradox* doomed Frege's version of logicism. And many now regard Kurt Gödel's celebrated *incompleteness theorems* as dooming all other versions too (although philosophers Crispin Wright and Bob Hale proposed a *neologicist* revival of Frege's project). But, in another way, Frege's project was an unprecedented success. It led to a monumental explosion of developments in logic and its applications (most notably, the digital computer). Many of these developments–like Frege's logicism–were fueled by integrating logic with its applications. So, it is ironic that one of the many insights to emerge was the importance of cleanly distinguishing between logic and its applications. It's doubly ironic that this insight is often ignored when teaching logic.

Our approach will be to teach logic in a way that respects this insight. We will introduce notions of logical form in the abstract, without regard to their applications. Only then will we consider some prominent applications. This approach requires patience. But the patience will be rewarded.

1.1 Form

Logic concerns **form**. But there is no one-size-fits-all answer to what form is. Still, we may begin to clarify what form is by contrasting it with what it is not. **Form** is opposed to **content**.

We are already familiar with the form/content distinction in various guises. One of these originated with Aristotle and construes content as *matter*. A table is not just its *matter*–its legs and top–scattered about. It is the legs and top in a certain arrangement, or *form*. And so too for chairs, cars, cellphones, and many other objects.

Construing content as matter is not limited to midsized objects. It scales up to the massive and down to the miniscule. For example, scaling up: A galaxy is not just its *matter*–stars, planets, and the

like–scattered across the universe. It is these phenomena in a certain gravitational clustering, or *form*. And so too for galaxy clusters and many other astronomical phenomena. For another example, scaling down: A water molecule is not just its *matter*–two hydrogen atoms and an oxygen atom–flitting about. It is these atoms bonded in a certain structure, or *form*. And so too for other molecules.

Some may even construe content as *mental*. On their view, a person (or at least their personality) is not just a jumble of thoughts, desires, emotions, memories, and experiences. It is all of these in a connected or continuous unity, or *form*.

We are also familiar with more abstract forms that may appear to be neither material nor mental. One is **musical form**. Consider the song *Tainted Love*. The original recording, by Gloria Jones, is in the style of Motown. The most famous recording, by Soft Cell, is New Wave synthpop. A more obscure recording, by Coil, is a bleak take on the 1980's AIDS crisis. These three recordings differ in timbre, instrumentation, mood, and more. But we may abstract away from these details–their *content*–and focus on the patterns of melody and rhythm–their *form*–that makes them all versions of the same song.

We are also familiar with **mathematical form**. To illustrate, notice how order does not matter for *addition* but does for *subtraction*:

$$5 + 7 = 7 + 5$$
$$5 - 7 \neq 7 - 5$$

These are instances of general facts. Abstracting from the specific numbers–their *content*–we see a pattern for sums–their *form*:

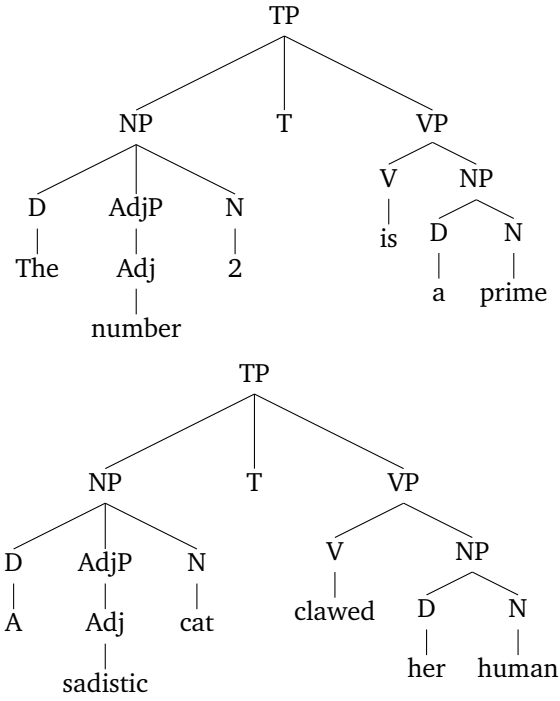
Commutative Law of Addition. $a + b = b + a$

Substraction is not commutative. It has no corresponding law.

Finally, we are familiar with **linguistic form**. Different sentences may share a grammatical form. For example, consider:

The number 2 is a prime.
A sadistic cat clawed her human.

We may abstract their differences–their *content*–and focus on their shared grammatical structure. Linguists represent this *form* as:



Each notion of form abstracts away from content irrelevant to a topic. Musical form abstracts from the color of instruments. Mathematical form abstracts from the numerals (Arabic vs. Roman). Grammatical form abstracts from font. What does *logical* form abstract away from?

The short answer is that logical form abstracts away from content itself. Not just musical, mathematical, or linguistic content. But content *in general*. The long answer is more nuanced. It will take awhile to refine properly. This book itself is only part of the refinement.

It begins by crafting an artificial language. We craft its **syntax** so that *only* forms we are interested are grammatical. Next, we craft a **semantics** that determines how form and content interact. Then we craft a system of **derivation** for transforming forms into others. Details irrelevant to form are inexpressible in our artificial language. This is by design. It makes the artificial language a **formal language**. And that makes it an ideal tool for studying form and its applications.

The most prominent application is to whatever may be true or false (statements, representations, propositions). The focus is only

on their form *as opposed to* their content. This is both limiting and powerful. It is limiting in that logic *alone* cannot deliver content-specific results. But it is powerful in that logic can apply to anything whatsover that may be true or false, regardless of its content. This gives our application of formal logic enormous generality. Many claims we make–whether in philosophy, science, religion, or elsewhere–are the sorts of claims that may be true or false. Our notion of form can apply to them all. These claims can also be combined in various ways. Our notion of form will apply to sets of such claims. And some of these sets will be of special interest because they represent evidence or justification for some other claim. Our notion of form will also apply to sets like these. In short, our abstract notion of form applies generally to anything that may be true or false.

1.2 Resources

Our exploration of form requires some conceptual resources.

1.2.1 Use and mention

When studying a formal language, we usually *mention* its symbols rather than *use* them. This requires a general distinction between *using* and *mentioning* words. To illustrate, contrast how we may speak about Seattle *the city* or Seattle *the word*. Consider:

Seattle is rainy.

This is about a city. We *use* the name 'Seattle' to refer to the city to say that it is rainy. That's true. Now, consider:

'Seattle' is the Anglicization of the Duwamish name 'Si'ahl'.

This is about a word. We *mention* the name 'Seattle' to speak of its etymology. Again, that's true. But consider:

'Seattle' is rainy.

This is about a word. We *mention* the name 'Seattle' to say it is rainy. That's false. Words aren't rainy. Now, consider:

Seattle is the Anglicization of the Duwamish name 'Si'ahl'.

This is about a city. We *use* the name ‘Seattle’ to make a claim about the city’s etymology. That’s false. Cities aren’t words.

It is vital to distinguish between whether a word is used or mentioned. Single quotes (‘ ’) signal the *mention* (as opposed to the *use*) of a word. Single quotes can be iterated. Some theoretical purposes require naming names. Consider:

“Seattle” names ‘Seattle’.

The double quote does not report speech. Instead, “Seattle” names the name ‘Seattle’. Generally, single quotes do not report speech. Double quotes do. (To quote Yoda, “The greatest teacher, failure is.”)

1.2.2 Object language and metalanguage

We’ve used the same language (English) to mention words in *it*. But we can also use one language to mention words in *another* language.

This is familiar from translation. When learning a new language, we often use a language we already know to mention words in a language we are learning. For example, you can learn what a German sentence says by using English to mention it and say what it means:

‘Es regnet’ means that it is raining.

A more indirect way mentions the German sentence and its English translation and says that they have the same meaning:

‘It is raining’ and ‘Es regnet’ have the same meaning.

These examples illustrate how the language we are using may differ from the language we are mentioning.

The language we talk about, or **mention**, is the OBJECT LANGUAGE. The language we **use** to talk about it is the METALANGUAGE. The object language may, or may not, be part of the metalanguage.

The general point is that, whenever we want to talk in English about some expression of a formal language, we must indicate that we are *mentioning* the expression, rather than *using* it. We can signal this with single quotes, or we can adopt some similar convention, such as placing it centrally in the page.

We can be sloppier for formal languages. Their expressions have

no content. They cannot be *used*, only *mentioned*. So, we may just read the expressions of our formal languages *as if* they were enclosed in single quotes. This is harmless because we never use those expressions. And it is practical because it declutters the page.

1.2.3 Metavariables

It is useful to have a device for talking about *arbitrary* expressions of a formal language. We use bold-faced **metavariables**:

> A METAVARIABLE is a symbol (‘**A**’,‘**B**’,‘**C**’,...) in a metalanguage that is used to refer to arbitrary expressions of a specified object language.

Metavariables can be used to make generalizations about the object language. To illustrate, we may want to say that every English sentence (of a sort) can be used to generate a new one by prefixing ‘Surely’ to it. So, for example, each sentence on the right is generated from the one to its left by this prefixing:

$$\begin{array}{rcl} \text{Pigs fly.} & \Longrightarrow & \text{Surely, pigs fly.} \\ \text{Fido barks.} & \Longrightarrow & \text{Surely, Fido barks.} \\ \text{Surely, Fido barks.} & \Longrightarrow & \text{Surely, surely, Fido barks.} \end{array}$$

This could continue indefinitely. But it needn’t. We can more succinctly express what we want by using a metavariable:

If **S** is an English sentence, then ‘Surely, **S**’ is too.

But this is wrong. It entails that this is an English sentence:

Surely, **S**

But ‘**S**’ is not an expression of English. It stands for an arbitrary sentence of English without being one of them (Compare: in algebra, the variable x stands for an arbitrary number without being one of them.) Generally, we may *use* a metavariable in a metalanguage to *mention* expressions of an object language that *lacks* the metavariable.

We want to interpret ‘Surely, **S**’ as: “the result of prefixing ‘Surely’ to an arbitrary English sentence produces an English sentence”. But single quotes *mention* what’s in between them. So, we cannot interpret them correctly without also mentioning the metavariable itself.

The philosopher W.V. O. Quine introduced some notation to help. He uses rectangular corner-quotes and interprets them in the desired way. So, our earlier statement becomes:

If **S** is an English sentence, then ⌜Surely, **S**⌝ is too.

This says what we want. In practice, context often makes clear what we want to say. So, we may often omit corner-quotes.

1.2.4 Sets and Sequences

We will often discuss collections of expressions in our formal language. We focus on two kinds: **sets** and **sequences**. A **set** merely groups its items together, without regard to order or repetition. A **sequence** groups its items in a way that is sensitive to order and repetition. There are rich theories of sets and sequences. Our purposes, however, only require familiarity with what follows.

A **set** is a collection of items, ignoring order and repetition. Sets have their own notation. We symbolize a set of items by enclosing them in curly brackets. For example, the set of items a, b, c is written as $\{a, b, c\}$.So, all of the following are the same set:

$$\{1,2,3\} \quad \{3,2,1\} \quad \{3,2,1,1,2,3\}$$

This notation lists a set's members. That may be infeasible. We may want to speak of the set of natural numbers $0, 1, 2, \ldots$ without listing them all. We may use uppercase Greek letters as names for sets: Γ (gamma), Δ (delta), and Σ (sigma). So, for example, we could let Γ be the set of electrons. (Mathematics already has a symbol '$\mathbb{N}$' for the set of natural numbers.) We also use notation for saying that an item x belongs to, or *is a member of*, a set Γ:

$$x \in \Gamma$$

and to write that x is not a member of set Γ we write:

$$x \notin \Gamma$$

Sets can be nested. This set nests the set $\{4,5\}$ inside another:

$$\{1,2,3,\{4,5\}\}$$

Its members are $1, 2, 3, \{4,5\}$. The set $\{4,5\}$ is a *member* of this set,

but not a *subset* of it.

> A set Γ is a **subset** of a set Δ ($\Gamma \subset \Delta$) if and only if every member of Γ is a member of Δ.

Neither 4 nor 5 are members of $\{1,2,3,\{4,5\}\}$ (although both are members of one of its members: $\{4,5\}$). By contrast, consider:

$$\{1,2,3,\{1,2,3\}\}$$

Its members are $1,2,3,\{1,2,3\}$. Set $\{1,2,3\}$ is a member *and* a subset.

It is important to distinguish between a set's *members* and its *subsets*. The identity of a set is determined by its *members* only:

> Set Γ = set $\Delta =_{def}$ for all x, $x \in \Gamma$ if and only if $x \in \Delta$.

The **union** of sets Γ, Δ ($\Gamma \cup \Delta$) is the set of every item that is a member of *at least one, or both* of Γ, Δ. Strictly speaking, when x is not a set, we must write its union with Γ as: '$\{x\} \cup \Gamma$'. But, as a more readable shorthand, we may also write: 'x, Γ'.

A **sequence** is a collection of its items that is sensitive to order and repetition. We use the notation $\langle a, b, c\rangle$ to denote the sequence of the items a, b, c *in that order*. Because order and repetition matter, none of the following are the same sequence:

$$\langle 1,2,3\rangle \quad \langle 3,2,1\rangle \quad \langle 3,2,1,1,2,3\rangle$$

We reuse the notation for set membership also for sequence membership. To say that x is a member of sequence Σ we write:

$$x \in \Sigma$$

and to write that x is not a member of set Σ we write:

$$x \notin \Sigma$$

The identity of a sequence is determined by its members, their order, and their repetition:

> Sequence $\langle a_0, a_1, \ldots\rangle$ = sequence $\langle b_0, b_1, \ldots\rangle =_{def}$ for all $i \geqslant 0$, $a_i = b_i$.

A sequence with a finite number n of items is an **n-tuple**. For example, $\langle 3,2,1\rangle$ is a 3-tuple and $\langle 0,1,2,3,2,1,0\rangle$ is a 7-tuple.

A set or a sequence may contain any number of items, finite or infinite. And they can nest. This is a set of 2-tuples:

$$\{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}$$

And a set or a sequence may even be *empty*: with *zero* members. There is a unique *empty set* which is symbolized as $\emptyset$. There is also a unique *empty sequence* which is symbolized as $\langle\rangle$. Although $\emptyset$ and $\langle\rangle$ are alike in that neither has members, they are not the same. The empty set $\emptyset$, like all sets, is *orderless*. The empty sequence $\langle\rangle$, like all sequences, is *ordered*. So, $\emptyset \neq \langle\rangle$.

1.2.5 Size, Minimum, and Maximum

The number of members in a set (sequence) is its **size**: $\#(\Gamma)$. So:

$$\begin{gathered}\#(\{1,2,3\}) = 3\\ \#(\{2,3,3,3,1\}) = 3\\ \#(\langle 2,3,3,3,1\rangle) = 5\end{gathered}$$

The size of the empty set $\emptyset$ (and empty sequence $\langle\rangle$) is 0.

We may ask whether a set (or sequence) Σ has a least or a greatest member. It may have both, one or the other, or neither. If it has a least member, that is its **minimum**: $\min(\Sigma)$. And if it has a greatest member, that is its **maximum**: $\max(\Sigma)$. For example:

$$\begin{gathered}\min(\{1,2,3\}) = 1\\ \max(\langle 2,3,3,3,1\rangle) = 3\end{gathered}$$

These can be nested, as in:

$$\max(\{\max(\{0,1\}), \min(\langle 2,3,4\rangle)\}) = 2$$

Order and repetition cannot affect the minimum or maximum. Calculating them therefore doesn't require distinguishing between sets or sequences. So, we may declutter our notation by omitting brackets:

$$\max(\max(0,1), \min(2,3,4)) = 2$$

The empty set $\emptyset$ (and empty sequence $\langle\rangle$) gets special treatment. It has no minimum or maximum because it has no members. But it has a *greatest lower bound*: 1. And it has a *least upper bound*: 0. So, we stipulate that: $\min(\emptyset) = 1$ and $\max(\emptyset) = 0$.

PART II

Sentential Form

CHAPTER 2
Syntax

This chapter focuses on the syntax of a formal language, SL, capturing sentential form.

2.1 Expressions

Simple expressions are symbols *not* constructed from others, while **complex expressions** are. We do not yet consider whether either are meaningful. We will when we get to their *semantics*. Our focus now is just on their grammar or *syntax*.

To illustrate, English's simple expressions include familiar symbols: letters ('a','b','c',...) and punctuation ('!','?','(',')',...). Any string of these forms a complex expression, such as:

Mike is chasing Bob!
Es regnet.
,,,, Klurb lLlL!???((

Not all of these are grammatical or meaningful *in English*. But all are still expressions in it. By contrast, none of these are:

¬ ∨ ∨∨
$\phi\iota\lambda o\sigma o\phi\iota\alpha$
⌊ The dog ate $\aleph_0$⌋

This is because they contain symbols that are *not* simple expressions *in English*.

Complex expressions of SL are made of its simple expressions. These are of three kinds: **elements**, **connectives**, and **punctuation**. Together, they compose SL's **alphabet**:

The ALPHABET OF SL consists in:

ELEMENTS	**Verum**	$\top$
	Falsum	$\bot$
	Letters	$A, B, ..., Z, ..., A_6, ..., Z_6, ...$
CONNECTIVES	**Negation**	$\neg$
	Disjunction	$\vee$
	Conjunction	$\wedge$
	Conditional	$\rightarrow$
	Biconditional	$\leftrightarrow$
PUNCTUATION		(,)

Subscripts ('$_0$', '$_{113}$') allow for infinite letters. An expression just strings together elements, connectives, and punctuation:

An SL EXPRESSION is any string from its alphabet.

These are all examples of expressions *in SL*:

$$(A \leftrightarrow B)$$
$$AAAAAAAAA_1A_2A_1AAAAAAAA$$
$$\neg)(\vee() \wedge (\neg\neg())((Q$$
$$SUNNO)))$$

But not all are *grammatical* or *syntactically well-formed* in SL.

2.2 Sentences

We now define SL's grammar. Our definition is *inductive*. An inductive definition specifies *base* items and *inductive* rules for generating more items from base or generated items. Here is an inductive definition for the grammatical expressions, or *sentences*, of SL:

A SENTENCE OF SL is generated by these rules:

BASE	Every element is a sentence.
INDUCTIVE	$\neg$**A** is a sentence if **A** is.
	(**A** $\wedge$ **B**) is a sentence if **A**,**B** are.
	(**A** $\vee$ **B**) is a sentence if **A**,**B** are.
	(**A** $\rightarrow$ **B**) is a sentence if **A**,**B** are.
	(**A** $\leftrightarrow$ **B**) is a sentence if **A**,**B** are.
CLOSURE	Nothing else is a sentence.

The BASE rule provides initial inputs for the INDUCTIVE rule, which

specifies how connectives generate sentences from the inputs. The CLOSURE rule says that these are the *only* way to generate sentences.

Our definition defines an infinite class of sentences. To illustrate, take element P. By BASE, P is a sentence. Put P for **A**. Then INDUCTIVE generates sentence $\neg P$. Put $\neg P$ for **A**. By INDUCTIVE, $\neg\neg P$ is a sentence. Repeating this infinitely expands SL's sentences. Putting P for **A** and $\neg P$ for **B** generates the left column and reversing the order generates the right column:

$$\begin{array}{ll} (P \wedge \neg P) & (\neg P \wedge P) \\ (P \vee \neg P) & (\neg P \vee P) \\ (P \rightarrow \neg P) & (\neg P \rightarrow P) \\ (P \leftrightarrow \neg P) & (\neg P \leftrightarrow P) \end{array}$$

We can also input P, or $\neg P$, "twice over" into each of **A**,**B**:

$$\begin{array}{ll} (P \wedge P) & (\neg P \wedge \neg P) \\ (P \vee P) & (\neg P \vee \neg P) \\ (P \rightarrow P) & (\neg P \rightarrow \neg P) \\ (P \leftrightarrow P) & (\neg P \leftrightarrow \neg P) \end{array}$$

Generated sentences can be input again into rules to generate more:

$$((P \wedge P) \vee (P \leftrightarrow \neg P))$$
$$(P \rightarrow \neg(\neg\neg(P \wedge P) \wedge (P \vee \neg P)))$$
$$\neg((\neg P \rightarrow (P \vee (\neg P \wedge P))) \vee (P \rightarrow \neg(\neg\neg(P \wedge P) \wedge (P \vee \neg P))))$$

We have an infinite supply of other letters. So, we can do for them what was done for P. And we can mix and match:

$$((B \wedge A) \vee (G \leftrightarrow \neg H))$$
$$(C \rightarrow \neg\neg\neg((P \wedge P) \wedge (Q \vee \neg A)))$$
$$\neg((A \leftrightarrow (\neg D \rightarrow P_0)) \vee ((B_1 \wedge P_{113}) \vee (Q \leftrightarrow \neg P)))$$

The grammatical rules inductively define these, and infinitely more, sentences of SL. Each is generated from finite elements plus finite applications of connectives. While some are long, none is infinite.

Whether or not a sentence has connectives depends on how it is generated. A sentence generated by BASE has none. It is **atomic**:

An ATOMIC SENTENCE is a sentence without connectives.

Each element is an atomic sentence. By contrast, a sentence gener-

ated by INDUCTIVE has at least one connective. It is **complex**:

> A COMPLEX SENTENCE is a sentence with connective(s).

A complex sentence joins other sentences with its connective(s). So, it has other sentences, or *subsentences*, as parts. An atomic sentence has no connectives, so no subsentences. Every sentence is atomic or complex, but never both.

How a sentence is generated from its atomic sentences is its **ancestry**. For example, the ancestry of $(A \to (\neg B \to A))$ is:

1. A, B are sentences	(BASE)
2. $\neg B$ is a sentence	(INDUCTIVE on 1)
3. $(\neg B \to A)$ is a sentence	(INDUCTIVE on 1,2)
4. $(A \to (\neg B \to A))$ is a sentence	(INDUCTIVE on 1,3)

Ancestry is *unique*: a sentence can *only* be generated one way. This important fact is often called **unique readability**.

Unique readability ensures that some connective is used *last* to generate a complex sentence. This is its **main connective**:

> The MAIN CONNECTIVE of a complex sentence is the connective used last in generating it.

A complex sentence may have many connectives, even the same connective many times. But there is always a unique main connective. For example, the ancestry above used $\to$ twice. Its last (leftmost) occurrence is its main connective. No atomic sentence has a main connective (because they have none). There is some standard terminology for complex sentences:

- $\neg\mathbf{A}$ is a NEGATION, and $\mathbf{A}$ is its NEGATUM
- $\mathbf{A} \wedge \mathbf{B}$ is a CONJUNCTION, and $\mathbf{A}, \mathbf{B}$ are its CONJUNCTS
  $\mathbf{A}$ is its LEFT CONJUNCT, $\mathbf{B}$ is its RIGHT CONJUNCT
- $\mathbf{A} \vee \mathbf{B}$ is a DISJUNCTION, and $\mathbf{A}, \mathbf{B}$ are its DISJUNCTS
  $\mathbf{A}$ is its LEFT DISJUNCT, $\mathbf{B}$ is its RIGHT DISJUNCT
- $\mathbf{A} \to \mathbf{B}$ is a CONDITIONAL
  $\mathbf{A}$ is its ANTECEDENT, $\mathbf{B}$ is its CONSEQUENT
- $\mathbf{A} \leftrightarrow \mathbf{B}$ is a BICONDITIONAL

2.3 Scope

A connective may, or may not, apply to an entire sentence. How much it applies to is its **scope**:

> The SCOPE of a connective in a sentence is the sentence that has it as its main connective.

To illustrate, consider the sentence from above:

$$(A \to (\neg B \to A))$$

Its main connective is the leftmost $\to$. Its scope is the *entire* sentence. The scope of the rightmost $\to$ is the subsentence:

$$(\neg B \to A)$$

And the scope of the $\neg$ is the subsentence:

$$\neg B$$

Scope ambiguities are cases where a connective's scope is unclear. An analogy from mathematics illustrates the phenomenon. What is the answer to this problem?

$$5 - 4 + 1$$

It depends. The answer is 2 if the scope of '+' is the whole problem: it is an addition problem that calculates $5 - 4$ first and then adds 1. But the answer is 0 if the scope of '–' is the whole problem: it is a substraction problem that calculates $4 + 1$ first and then subtracts 5. The original question is ambiguous: the scopes of '–' and '+' are unclear. There are various strategies for disambiguating. One policy is to calculate addition before subtraction. (This is captured in the acronym *PEMDAS*.) Another, syntactic strategy uses brackets to signal the order of calculations:

$$(5 - 4) + 1 = 2$$
$$5 - (4 + 1) = 0$$

Which strategy is used is less important than *that some* is used.

Our syntactic strategy avoids scope ambiguities with brackets. To illustrate, suppose we remove all brackets from our example above:

$$A \to \neg B \to A$$

This can be read as ambiguous between these sentences:

$$(A \to (\neg B \to A))$$
$$((A \to \neg B) \to A)$$
$$(A \to \neg(B \to A))$$

The bracketless expression itself is *not* uniquely readable. It cannot be generated by the rules for generating a sentence of SL. So, it is *not* a sentence of SL. Adding brackets is essential to producing an unambiguous sentence.

But we may often omit brackets to improve readability, at least when there is no risk of ambiguity.

We may always omit the *outermost* brackets. Strictly speaking, this gives an expression that is *not* a sentence of SL. For example, $\neg B \to A$ is not a sentence of SL. But it is more readable than $(\neg B \to A)$. We must reinsert the brackets when we embed a sentence into another. Not doing so risks introducing the syntactic ambiguities that we wish to avoid.

We also allow for *stylistic variants* of brackets. Specifically, we may use square brackets, '[' and ']', too. So, $(\neg B \to B)$ and $[\neg B \to B]$ are stylistic variants. Square brackets aid readability. The eyesore:

$$(((P \to Q) \vee (R \to S)) \wedge (T \vee U))$$

becomes the more readable:

$$\big[(P \to Q) \vee (R \to S)\big] \wedge (T \vee U)$$

These conventions have two further benefits. One is that they make it easier to identify a sentence's main connective. Another is that they make it easier to identify the scopes of all of its connectives.

CHAPTER 3
Semantics

This chapter focuses on the formal semantics of SL.

A **formal semantics** shows how forms captured by a formal language structure its content. This requires representing content, but only those aspects of it structured by form. The rest is irrelevant to formal semantics and may be ignored. These abstract representations of content are **semantic values**: *placeholders* for the content structured by the forms of our formal language.

A formal semantics determines the semantic values of its expressions. This is not done one by one, but **compositionally**. The semantic values of a select basis of expressions are specified *directly* and then rules are used to generate *indirectly* the semantic values for the rest. So, specifying a formal semantics requires specifying how the basic semantic values are directly determined (3.1) and the rules for indirectly determining the rest from them (3.2).

3.1 Valuations

The forms SL captures are those of the connectives. Letters are placeholders for content. A formal semantics assigns semantic values to letters. These are input into rules for connectives that output semantic values to complex sentences.

We need a pair of semantic values. They could be anything, as long as one is "greater" than the other. We use 0 and 1. ('0' and '1' are expressions of our metalanguage, *not* SL.)

The semantic values of letters are specified by a **valuation**:

> A VALUATION is an assignment of semantic values to letters.

There is an infinity of SL sentences. So too for valuations: one for each permutation of 1's and 0's. That's too many to display. But we can *partially* display some valuations $v_0, v_1, ..., v_*$ as:

	A	A_0	A_1	...	B	B_0	B_1	...	Z	Z_0	Z_1	...
v_0 :	0	0	0	...	0	0	0	...	0	0	0	...
v_1 :	1	0	0	...	0	0	0	...	0	0	0	...
v_2 :	1	1	0	...	0	0	0	...	0	0	0	...
$\vdots$				$\vdots$				$\vdots$				$\vdots$
v_* :	1	1	1	...	1	1	1	...	1	1	1	...

For example, valuation v_0 assigns 0 to A_0 (or: $v_0(A_0) = 0$). And valuation v_2 assigns 1 to A_0 (or: $v_2(A_0) = 1$). Each valuation assigns a distinct permutation of 1's and 0's to all letters.

3.2 Formal Semantics

We add new notation to abbreviate 'the semantic value of $\cdot$ for valuation v' (where $\cdot$ stands for a particular expression):

$$[\![\cdot]\!]_v$$

This notation extends to a set or sequence of expressions. We write $[\![\Gamma]\!]_v$ when Γ is a set $\{e_0, e_1, \ldots\}$ of expressions as a shorthand for:

$$\{[\![e_0]\!]_v, [\![e_1]\!]_v, \ldots\}$$

We omit the subscript v when the valuation is clear.

The formal semantics for SL specifies $[\![\cdot]\!]_v$ for all sentences in stages. Atoms are assigned semantic values *directly* by a valuation (A1-A3). Complex sentences are assigned semantic values *indirectly*: compositional rules for the connectives assign their semantic value on the basis of those of their parts (C1-C5).

The SEMANTIC VALUE $[\![\cdot]\!]_v$ for valuation v is:

A1	$[\![\top]\!]_v = 1$
A2	$[\![\bot]\!]_v = 0$
A3	$[\![\mathbf{A}]\!]_v = v(\mathbf{A})$ (for letter $\mathbf{A}$)
C1	$[\![\neg\mathbf{A}]\!]_v = 1 - [\![\mathbf{A}]\!]_v$
C2	$[\![\mathbf{A} \vee \mathbf{B}]\!]_v = \max([\![\mathbf{A}]\!]_v, [\![\mathbf{B}]\!]_v)$
C3	$[\![\mathbf{A} \wedge \mathbf{B}]\!]_v = \min([\![\mathbf{A}]\!]_v, [\![\mathbf{B}]\!]_v)$
C4	$[\![\mathbf{A} \rightarrow \mathbf{B}]\!]_v = \max([\![\neg\mathbf{A}]\!]_v, [\![\mathbf{B}]\!]_v)$
C5	$[\![\mathbf{A} \leftrightarrow \mathbf{B}]\!]_v = \min([\![\mathbf{A} \rightarrow \mathbf{B}]\!]_v, [\![\mathbf{B} \rightarrow \mathbf{A}]\!]_v)$

A1-A2 assign fixed semantic values to $\top$ and $\bot$. A3 assigns semantic values to letters. C1-C5 assign semantic values to complex sentences of connective from the semantic values of its parts. This traces a sentence's ancestry back to its atoms (2.2). To illustrate:

$$[\![A \to \neg(B \vee C)]\!]$$

This is a conditional. So, applying rule C4 gives:

$$\max([\![\neg A]\!], [\![\neg(B \vee C)]\!])$$

Applying rule C1 to both parts gives:

$$\max(1 - [\![A]\!], 1 - [\![B \vee C]\!])$$

And applying rule C2 gives:

$$\max(1 - [\![A]\!], 1 - \max([\![B]\!], [\![C]\!]))$$

This shows how the semantic value of the *complex* sentence may be calculated from the semantic values of its *atoms*. This works because SL's connectives are **compositional**:

> A connective is COMPOSITONAL iff the semantic value of a sentence with it as its main connective is uniquely determined by the semantic value(s) of its subsentence(s).

3.3 Semantic Tables

A semantic rule is neatly displayed by a **semantic table**: a chart with a sentence on the right, its atoms on the left, and rows for valuations. Each sentence is generated from finite atoms. So, a row only needs to represent the *part* of the valuation concerning them.

The semantic tables for A1-A2 show how the semantic values of $\top$ and $\bot$ are assigned *regardless* of the valuation:

$\top$	$\top$
1	**1**

$\bot$	$\bot$
0	**0**

The semantic table for a letter shows how a valuation assigns its semantic value, just as A3 says:

A	A
1	1
0	0

And the semantic tables for c1-c5 are:

A	¬A
1	0
0	1

A	B	A∨B
1	1	1
1	0	1
0	1	1
0	0	0

A	B	A∧B
1	1	1
1	0	0
0	1	0
0	0	0

A	B	A→B
1	1	1
1	0	0
0	1	1
0	0	1

A	B	A↔B
1	1	1
1	0	0
0	1	0
0	0	1

Semantic tables can also show how the semantic value of a complex sentence is determined by its parts and connectives. Consider:

$$(A \vee B) \rightarrow (A \leftrightarrow \neg(B \wedge A))$$

We will build a semantic table in stages:

First, make two columns, with atoms on the left and the complex sentence on the right:

A	B	$(A \vee B) \rightarrow (A \leftrightarrow \neg(B \wedge A))$

Second, list the valuations, one per row, in any order:

A	B	$(A \vee B) \rightarrow (A \leftrightarrow \neg(B \wedge A))$
1	1	
1	0	
0	1	
0	0	

Third, copy the columns under each of the atoms on the left over to the columns under each of the atoms on the right:

A	B	$(A \vee B) \rightarrow (A \leftrightarrow \neg(B \wedge A))$						
1	1	1	1	1		1	1	
1	0	1	0	1		0	1	
0	1	0	1	0		1	0	
0	0	0	0	0		0	0	

Finally, calculate using the semantic rules, working from the innermost connectives outward to the main connective:

A	B	$(A \vee B) \rightarrow (A \leftrightarrow \neg(B \wedge A))$						
1	1	1	1	1		**1**	**1**	1
1	0	1	0	1		**0**	**0**	1
0	1	0	1	0		**1**	**0**	0
0	0	0	0	0		**0**	**0**	0

A	B	$(A \vee B) \rightarrow (A \leftrightarrow \neg(B \wedge A))$						
1	1	1	1	1	**0**	1	1	1
1	0	1	0	1	**1**	0	0	1
0	1	0	1	0	**1**	1	0	0
0	0	0	0	0	**1**	0	0	0

A	B	$(A \vee B) \rightarrow (A \leftrightarrow \neg(B \wedge A))$								
1	1	1	1	**1**	**0**	**0**	1	1	1
1	0	1	0	**1**	**1**	**1**	0	0	1
0	1	0	1	**0**	**0**	**1**	1	0	0
0	0	0	0	**0**	**0**	**1**	0	0	0

A	B	$(A \vee B) \rightarrow (A \leftrightarrow \neg(B \wedge A))$								
1	1	**1**	**1**	**1**	1	0	0	1	1	1
1	0	**1**	**1**	**0**	1	1	1	0	0	1
0	1	**0**	**1**	**1**	0	0	1	1	0	0
0	0	**0**	**0**	**0**	0	0	1	0	0	0

A	B	$(A \vee B) \rightarrow (A \leftrightarrow \neg(B \wedge A))$										
1	1	1	1	1	**0**	1	0	0	1	1	1
1	0	1	1	0	**1**	1	1	1	0	0	1
0	1	0	1	1	**0**	0	0	1	1	0	0
0	0	0	0	0	**1**	0	0	1	0	0	0

More atoms mean more rows. A sentence with n atoms needs 2^n rows. For example, $A \rightarrow \neg(B \vee C)$ needs $2^3 = 8$ rows:

A	B	C	$A \rightarrow \neg (B \vee C)$
1	1	1	1 **0** 0 1 1 1
1	1	0	1 **0** 0 1 1 0
1	0	1	1 **0** 0 0 1 1
1	0	0	1 **1** 1 0 0 0
0	1	1	0 **1** 0 1 1 1
0	1	0	0 **1** 0 1 1 0
0	0	1	0 **1** 0 0 1 1
0	0	0	0 **1** 1 0 0 0

3.4 Semantic Concepts

We define **entailment** and other semantic concepts with it.

3.4.1 Entailment

Entailment is a concept that applies to arguments. An argument is *not* a quarrel. It is a set of **premises** and a **conclusion**:

> An ARGUMENT is a set of sentences Γ (the PREMISES) and a sentence $\mathbf{C}$ (the CONCLUSION).

There can be any number of premises, even zero. We symbolize an argument from premises Γ to conclusion $\mathbf{C}$ as:

$$\Gamma \therefore \mathbf{C}$$

The symbol '$\therefore$' is pronounced "therefore". **Entailment** is defined for arguments, so understood:

> Γ ENTAILS $\mathbf{C}$ iff there is no v where $\min(\llbracket \Gamma \rrbracket_v) > \llbracket \mathbf{C} \rrbracket_v$

Entailment is symbolized as '$\models$' (*double turnstile*). When Γ entails $\mathbf{C}$, we write $\Gamma \models \mathbf{C}$ and say that $\mathbf{C}$ is a **consequence** of Γ. When Γ does *not* entail $\mathbf{C}$, we write $\Gamma \not\models \mathbf{C}$.

To illustrate, consider argument $\{A, A \rightarrow C\} \therefore C$. Do the premises entail the conclusion? They do only if:

$$\text{There is no } v \text{ where } \min(\llbracket A \rrbracket_v, \llbracket A \rightarrow C \rrbracket_v) > \llbracket C \rrbracket_v$$

We must check whether a valuation can assign a *lower* semantic value to the conclusion than to the minimum semantic value assigned to the

premises. It can only if it assigns the conclusion 0 and all the premises 1. In particular, it must assign 1 to A and 1 to $A \to C$. By rule c4, $A \to C$ is assigned to 1 only if the maximum semantic value of $\neg A$ and C is 1. By rule c1, $\neg A$ is assigned 0 because A is assigned 1. But then neither can be assigned 0. So, in particular, C is assigned 1. If so, then the conclusion *cannot* be lower than the minimum semantic value assigned to the premises. This shows that $\{A, A \to C\} \vDash C$.

For another example, consider argument $\{C, A \to C\} \therefore A$. Do the premises entail the conclusion? They do only if:

$$\text{There is no } v \text{ where } \min([\![C]\!]_v, [\![A \to C]\!]_v) > [\![A]\!]_v$$

Consider a valuation that assigns 0 to A and 1 to C. By rule c4, it assigns 1 to $A \to C$. The premises are all assigned 1. So, their minimum is 1. That is greater than the semantic value assigned to the conclusion. So, $\{C, A \to C\} \nvDash A$.

Semantic tables can encode the calculations. Add a column for each premise followed by one for the conclusion. (For the empty set of premises, just add the conclusion.)

Entailment: show that *no* row of the semantic table assigns 1 to all premises but 0 to the conclusion. This semantic table verifies that $\{A, A \to C\} \vDash C$:

A	C	A	$A \to C$	C
1	1	1	1 1 1	1
1	0	1	1 0 0	0
0	1	0	0 1 1	1
0	0	0	0 1 0	0

No row assigns 0 to C and 1 to A and $A \to C$.

***Non*entailment**: show that *some* row does so. This semantic table verifies that $\{C, A \to C\} \nvDash A$:

A	C	C	$A \to C$	A
1	1	1	1 1 1	1
1	0	0	1 0 0	1
0	1	1	0 1 1	0
0	0	0	0 1 0	0

The third row assigns 1 to all premsies but 0 to the conclusion.

Entailment has an important property: **monotonicity**:

MONOTONICITY. If $\Gamma \models \mathbf{C}$, then $\mathbf{A}, \Gamma \models \mathbf{C}$ (for any $\mathbf{A}$).

This means that if a sentence is a consequence of a set, then it is also a consequence of an arbitrary expansion of that set.

3.4.2 Validity (Tautology)

A **validity** is a consequence of the empty set ($\emptyset$) of premises:

$\mathbf{A}$ is a VALIDITY (or TAUTOLOGY) iff $\emptyset \models \mathbf{A}$ (or: $\models \mathbf{A}$).

Monotonicity implies that a validity is a consequence of all expansions to the empty set, so of *all* sets of premises. That means each valuation assigns it 1 *no matter what* the premises are assigned. So, validity may also be defined as:

$\mathbf{A}$ is a VALIDITY iff $[\![\mathbf{A}]\!]_v = 1$ for all v.

Validities of SL are often called *tautologies*. Others logics, such as ML and FOL recognize validities that are *not* tautologies.

Validity (**tautology**): show that *every* row of a sentence's semantic table assigns it 1. This works because a tautology is a consequence of the empty set of premises. Consider $A \to (B \to A)$:

A	B	$A \to (B \to A)$
1	1	1 **1** 1 1 1
1	0	1 **1** 0 1 1
0	1	0 **1** 1 0 0
0	0	0 **1** 0 1 0

Every row assigns 1 to $A \to (B \to A)$. So, it is a validity.

Non*validity** (non*tautology**): show that *some* row of a sentence's semantic table assigns it 0. Consider $A \to C$:

A	C	$A \to C$
1	1	1 **1** 1
1	0	1 **0** 0
0	1	0 **1** 1
0	0	0 **1** 0

The second rows assigns 0 to $A \to C$. So, it is not a validity.

3.4.3 Contradiction

If $\bot$ is a consequence of a sentence, then it is a contradiction.

A is a CONTRADICTION iff $\mathbf{A} \vDash \bot$.

A sentence entails $\bot$ only if each valuation assigns it 0. So, an equivalent definition of contradiction is:

A is a CONTRADICTION iff $[\![\mathbf{A}]\!]_v = 0$ for all v.

Because $\bot$ is always assigned 0, it is a contradiction.

Contradiction: show that *no* row of a sentence's semantic table assigns it 1. Consider $A \wedge (B \wedge \neg A)$:

A	B	$A \wedge (B \wedge \neg A)$
1	1	1 0 1 0 0 1
1	0	1 0 0 0 0 1
0	1	0 0 1 1 1 0
0	0	0 0 0 0 1 0

Every row assigns 0 to $A \wedge (B \wedge \neg A)$. So, it is a contradiction.

***Non*contradiction**: show that *some* row of a sentence's semantic table assigns it 1. Recall $A \to C$. It was assigned 1 on all but its second row. So, it is not a contradiction.

3.4.4 Contingency

A **contingency** is neither a validity nor a contradiction:

A is a CONTINGENCY iff **A** is not a validity or a contradiction.

Contingency: show that a sentence is neither a validity nor a contradiction. Use the tests above.

***Non*contingency**: show that a sentence is a validity or a contradiction. Again, use the tests above.

3.4.5 Equivalence

Equivalent sentences have the same consequences:

> $\mathbf{A}, \mathbf{B}$ are EQUIVALENT iff $\mathbf{A} \vDash \mathbf{B}$ and $\mathbf{B} \vDash \mathbf{A}$.

A valuation that assigned 1 to one sentence and 0 to the other would show that the second was *not* a consequence of the first. So, an equivalent definition of equivalence is:

> $\mathbf{A}, \mathbf{B}$ are EQUIVALENT iff $[\![\mathbf{A}]\!]_v = [\![\mathbf{B}]\!]_v$ for all v.

Equivalence: check that *every* row of a joint semantic table assigns both sentences the same semantic value. Consider the pair $A \to C$ and $\neg A \vee C$:

A	C	$A \to C$	$\neg A \vee C$
1	1	1 **1** 1	1 0 **1** 1
1	0	1 **0** 0	1 0 **0** 0
0	1	0 **1** 1	0 1 **1** 1
0	0	0 **1** 0	0 1 **1** 0

Each row assigns the pair the same semantic value. So, they are equivalent.

***In*equivalence**: show that *some* row of a joint semantic table assigns its sentences different semantic values. Consider $A \to C$ and $\neg(A \vee C)$:

A	C	$A \to C$	$\neg(A \vee C)$
1	1	1 **1** 1	**0** 0 1 1
1	0	1 **0** 0	**1** 0 0 0
0	1	0 **1** 1	**0** 1 1 1
0	0	0 **1** 0	**0** 1 1 0

No row assigns the same semantic value. So, the pair is inequivalent.

3.4.6 Satisfiability

A set of sentences is **unsatisfiable** if it entails a contradiction ($\bot$):

> Γ is UNSATISFIABLE iff $\Gamma \vDash \bot$.

If the set is unsatisfiable, then every valuation must assign 0 to *some* member of the set. So, an equivalent definition is:

Γ is UNSATISFIABLE iff $\min([\![\Gamma]\!]_v) = 0$ for all v.

A contradiction is a special case: an unsatisfiable set of *one*.

A set of sentences is **satisfiable** if it entails no contradiction:

Γ is SATISFIABLE iff $\Gamma \nvDash \bot$.

Not all valuations must assign 0 to some member of the set. The empty set is satisfiable: it has no members to assign 0 to.

Satisfiability: show that *some* row of a joint semantic table assigns 1 to *all* members of the set. Consider $\{A, \neg B, C\}$:

A	B	C	A	$\neg B$	C
1	1	1	1	0 1	1
1	0	1	1	1 0	1
1	1	0	1	0 1	0
1	0	0	1	1 0	0
0	1	1	0	0 1	1
0	0	1	0	1 0	1
0	1	0	0	0 1	0
0	0	0	0	1 0	0

The second row assigns 1 to them all. So, the set is satisfiable.

Unsatisfiability: show that *no* row of a joint semantic table assigns 1 to all of its sentences. Consider $\{A, A \rightarrow C, \neg C\}$:

A	C	A	$A \rightarrow C$	$\neg C$
1	1	1	1 1 1	0 1
1	0	1	1 0 0	1 0
0	1	0	0 1 1	0 1
0	0	0	0 1 0	1 0

No row assigns 1 to them all. So, this set is unsatisfiable.

3.5 Decidability

Semantic tables give a method for testing whether any semantic concept applies. This method is especially powerful because it is **decidable**: it is possible to test whether any semantic concept applies by constructing a *finitely* large semantic table.

Decidability means that computers can be programmed to perform these tests. The tests would work *in principle*. The only limitations are *practical*: computing power, memory, and so on.

These practical limitations mean the method is often infeasible. Consider the following argument:

$$A \therefore \neg A \vee (\neg A \rightarrow (A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow A_4))))$$

Does $A \vDash \neg A \vee (\neg A \rightarrow (A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow A_4))))$? Is the conclusion a consequence of the premise? We check by constructing a semantic table. The argument contains 5 atoms. So, the table needs $2^5 = 32$ rows. A person could calculate this table. It'd just take patient brute force, no ingenuity or skill. But the calculations are tedious and error-prone. Just the sort of task for a computer.

But some cases are infeasible even for computers. Consider another argument whose conclusion extends the pattern from the previous conclusion:

$$A \therefore \neg A \vee (\neg A \rightarrow (A_1 \rightarrow (A_2 \rightarrow ...(A_{98} \rightarrow A_{99})...)$$

Is its conclusion a consequence of its premise? That is, does $A \vDash \neg A \vee (\neg A \rightarrow (A_1 \rightarrow (A_2 \rightarrow ...(A_{98} \rightarrow A_{99})...)$? We test for this by constructing a semantic table. The argument contains 100 atoms. So, the table needs 2^{100} rows. That's more than the nanoseconds since the Big Bang. We (and maybe the cosmos) won't survive long enough to see the result.

It'd be nice to complement the semantic method with another one...

CHAPTER 4

Derivations

This chapter focuses on derivations for SL.

4.1 Derivations

The idea of a derivation is to "link" the premises of an argument to its conclusion, but only when they entail it. This gives a *syntactic* alternative to our semantic methods. More precisely:

> A DERIVATION is a finite sequence of sentences from a set Γ of premises to a conclusion **C**, with each intermediate sentence licensed by a derivational rule.

We display derivations in *Fitch format*: premises go first above a horizontal bar (nothing goes if there are none), the conclusion is last, and the linking steps in between. Lines are numbered (beginning with 1). To illustrate:

1	premises
⋮	⋮
	—
m	linking steps
⋮	⋮
n	conclusion

A derivation for $A \therefore \neg A \vee (\neg A \to (A_1 \to (A_2 \to (A_3 \to A_4))))$ begins:

1	A
	—
?	⋮
??	$\neg A \vee (\neg A \to (A_1 \to (A_2 \to (A_3 \to A_4))))$

The linking steps depend on the **derivational rules** (4.2). But first we introduce **subderivations** (4.1.1) and **visibility** (4.1.2).

4.1.1 Subderivations

Derivations may be nested. A SUBDERIVATION is a derivation nested inside another. We display this as:

1 | premises
2 | ⋮
3 | | subpremise
4 | | ⋮
5 | | subconclusion
6 | conclusion

A vertical bar marks the nesting level. The second bar spans lines 3-5 to mark where the subderivation begins and ends.

Subderivations can themselves be nested. There's no limit to how deep the nesting can go. Again, vertical bars signal the nesting level. Breaks mark the closing and opening of a subderivation within a level. This derivation nests two subderivations within a subderivation:

1 | premises
2 | ⋮
3 | | subpremise
4 | | | subsubpremise
5 | | | ⋮
6 | | | subsubconclusion
7 | | | subsubpremise
8 | | | ⋮
9 | | | subsubconclusion
10 | | subconclusion
11 | conclusion

One subderivation spans lines 3-10, marked by a second bar. It has two subsubderivations: one spanning lines 4-6, another spanning lines 7-9. A third bar marks each. The break between lines 6-7 marks the *closing* of the first subsubderivation and the *opening* of the second. In general, bars and breaks track what's nested within what.

4.1.2 Visibility

A derivational rule can only apply to a line or a subderivation it can "see". This can be made more precise by introducing some new terms:

> A line's DEPTH is the number of its vertical bars.

To illustrate, line 1's depth is 1, line 9's is 3, and line 10's is 2. Next, we define "separation" between equally deep lines:

> Lines m, n are SEPARATED iff m's depth = n's depth and there is a break between m and n.

To illustrate, no pair of lines 1-6 is separated, nor is any pair of lines 7-9. But lines 6 and 7 are separated, as are 5 and 9.

These notions help define what it is for a *line* or a *subderivation* to be **visible** to a line:

> Line m is VISIBLE to line n iff (i) $m < n$; (ii) m's depth $\leq n$'s depth; and (iii) m, n are not separated.

To illustrate, lines 1-5 are visible to line 6, but none of lines 6-11 (they are not *before* line 6). Lines 1-2 are visible to line 11, but none of lines 3-10 (they are *deeper*). And lines 7-8 are visible to line 9, but none of 4-6 (they are *separated*).

> A subderivation spanning lines i-j is VISIBLE to line n iff (i) $j < n$; (ii) i's depth = n's depth + 1; and (iii) i, j are not separated.

To illustrate, the subderivation spanning lines 3-10 is visible to line 11, but neither of the subsubderivations spanning lines 4-6 or 7-9 are.

4.2 Derivability

A **derivational system** specifies the derivational rules. A conclusion is **derivable** from premises if the rules can "link" them:

> **C** is DERIVABLE from Γ in SL iff there is a derivation from Γ to **C** using only SL's derivational rules.

Derivability has its own notation '$\vdash$' (*single turnstile*). We write $\Gamma \vdash \mathbf{C}$ when **C** is derivable from Γ and $\Gamma \nvdash \mathbf{C}$ when not.

A DERIVATIONAL RULE says how to extend a sequence of sentences in a derivation. Some rules only extend certain forms of sequences, while others extend any (even an empty sequence).

And some rules are bad. Consider a rule ★ that extends *any* sequence with **A** on line m with an *arbitrary* **B** (citing line m):

m	**A**
	B ★ m

This rule has unlimited power. It can derive anything from anything. So, it allows us to derive B from A:

1	A
2	B ★ 1

The unlimited power of ★ means that it also allows us to derive the negation of what we just derived:

3	$\neg B$ ★ 2

This makes ★ unsafe. Our semantic tests show that neither B nor $\neg B$ is a consequence of A. But ★ allows us to derive both from A. So, it is **invalid**. Derivations are a safe alternative to semantic tests only if they use **valid** rules: rules that *cannot* extend a sequence by a *nonconsequence* of anything earlier.

The simplest valid rule just reiterates something earlier:

m	**A**
	A R m

Because **A** $\vDash$ **A**, the rule is valid. But it only derives what we already have. There are many entailments that it alone cannot derive.

Our derivational system for SL balances the *safety* of valid rules with the *power* of making all entailments derivable. That is, it is both **sound** and **complete**:

SOUNDNESS. If $\Gamma \vdash \mathbf{C}$, then $\Gamma \vDash \mathbf{C}$.

COMPLETENESS. If $\Gamma \vDash \mathbf{C}$, then $\Gamma \vdash \mathbf{C}$.

SOUNDNESS says that *only* entailments of a set of premises are derivable from them. That's safety. COMPLETENESS says that *all* entailments of a set of premises are derivable from them. That's power.

It is a proven *metalogical* fact that our derivational system for SL is *both* sound *and* complete. This makes SL special. For it is also a proven metalogical fact that many logics more expressive than SL *cannot* be *both*: they are incomplete if sound.

4.2.1 Introduction and Elimination Rules

Our system's rules encode how a symbol (usually, a connective) is *introduced* or *eliminated*. An **introduction** (**I**) rule for a connective says when a sentence with it as its main connective may be derived. An **elimination** (**E**) rule for a connective says what may be *derived from* a sentence with it as its main connective. (A derived sentence may contain instances of an "eliminated" connective.)

4.2.1.1 Falsum

$\bot$I derives $\bot$ from a sentence and its negation:

m	**A**	
n	$\neg$**A**	
	$\bot$	$\bot$I m, n

$\bot$E derives any sentence from $\bot$:

m	$\bot$	
	A	$\bot$E m

The notation 'm' cites line m while 'm, n' cites both m and n. We may illustrate the rules by chaining them to show that $\{Q, \neg Q\} \vdash E \wedge F$:

1	Q	
2	$\neg Q$	
3	$\bot$	$\bot$I 1, 2
4	$E \wedge F$	$\bot$E 3

4.2.1.2 Conjunction

$\wedge$I derives a conjunction from its conjuncts:

$$\begin{array}{ll|l} m & \mathbf{A} & \\ n & \mathbf{B} & \\ \hline & \mathbf{A} \wedge \mathbf{B} & \wedge\text{I } m, n \end{array}$$

$\wedge$E derives a conjunct from a conjunction:

$$\begin{array}{ll|l} m & \mathbf{A} \wedge \mathbf{B} & \\ \hline & \mathbf{A} & \wedge\text{E } m \end{array} \qquad \begin{array}{ll|l} m & \mathbf{A} \wedge \mathbf{B} & \\ \hline & \mathbf{B} & \wedge\text{E } m \end{array}$$

Either conjunct may be derived. This is why we list each case.

We may illustrate the two rules by chaining them together to reorder the conjuncts of a conjunction:

$$\begin{array}{ll|l} 1 & E \wedge F & \\ \hline 2 & F & \wedge\text{E } 1 \\ 3 & E & \wedge\text{E } 1 \\ 4 & F \wedge E & \wedge\text{I } 2, 1 \end{array}$$

This shows that $\{E \wedge F\} \vdash F \wedge E$.

4.2.1.3 Negation

$\neg$I derives the negation of a sentence from a *subderivation* of falsum from the sentence:

$$\begin{array}{ll|l} i & \mathbf{A} & \\ \hline j & \bot & \\ \hline & \neg\mathbf{A} & \neg\text{I } i\text{–}j \end{array}$$

$\neg$E derives a sentence from a *subderivation* of falsum from the sentence's negation:

i	$\neg\mathbf{A}$	
j	$\bot$	
	$\mathbf{A}$	$\neg$E i–j

The notation 'i-j' cites the full *span* of lines from i to j. For example:

1	A	
2	$\neg A$	
3	$\bot$	$\bot$I 1, 2
4	$\neg\neg A$	$\neg$I 2–3

This shows that $\{A\} \vdash \neg\neg A$.

Some rules, like $\neg$I and $\neg$E, can begin a subderivation *at any point*. They may even begin a derivation from *zero* premises:

1	$A \wedge \neg A$	
2	A	$\wedge$E 1
3	$\neg A$	$\wedge$E 1
4	$\bot$	$\bot$I 2, 3
5	$\neg(A \wedge \neg A)$	$\neg$I 1–4

This shows that $\neg(A \wedge \neg A)$ is derivable from the *empty set* of premises. We write this as $\emptyset \vdash \neg(A \wedge \neg A)$, or $\vdash \neg(A \wedge \neg A)$.

The last derivation may be nested inside another to produce a longer, but equally correct, derivation of $\vdash \neg(A \wedge \neg A)$:

1	$\neg\neg(A \wedge \neg A)$	
2	$A \wedge \neg A$	
3	A	$\wedge$E 2
4	$\neg A$	$\wedge$E 2
5	$\bot$	$\bot$I 3, 4
6	$\neg(A \wedge \neg A)$	$\neg$I 2–5
7	$\bot$	$\bot$I 1, 6
8	$\neg(A \wedge \neg A)$	$\neg$E 1–7

4.2.1.4 Disjunction

∨I derives a disjunction of two sentences from either:

m	$\mathbf{A}$		m	$\mathbf{A}$	
	$\mathbf{A} \vee \mathbf{B}$	$\vee$I m		$\mathbf{B} \vee \mathbf{A}$	$\vee$I m

∨E derives a sentence from a disjunction when it is derivable from each of the disjuncts individually:

m	$\mathbf{A} \vee \mathbf{B}$	
i	$\mathbf{A}$	
j	$\mathbf{C}$	
k	$\mathbf{B}$	
l	$\mathbf{C}$	
	$\mathbf{C}$	$\vee$E m, i–j, k–l

This requires *two* distinct subderivations: one deriving **C** from **A**, another deriving **C** from **B**. We still add vertical bars to track depth. But we break the line between the first and second subderivations to show that they are *distinct*. To illustrate:

1	$D \vee E$	
2	D	
3	$D \vee (E \vee Q)$	$\vee$I 2
4	E	
5	$E \vee Q$	$\vee$I 4
6	$D \vee (E \vee Q)$	$\vee$I 5
7	$D \vee (E \vee Q)$	$\vee$E 1, 2–3, 4–6

This shows that $\{D \vee E\} \vdash D \vee (E \vee Q)$. Note that each subderivation is cited separately.

4.2.1.5 Conditional

→I derives a conditional from a subderivation deriving the consequent from the antecedent:

$$
\begin{array}{l|l|l}
i & \boxed{\mathbf{A}} & \\
j & \boxed{\mathbf{B}} & \\
 & \mathbf{A} \to \mathbf{B} & \to\text{I } i\text{–}j
\end{array}
$$

→E derives the consequent of a conditional from the conditional and its antecedent:

$$
\begin{array}{l|l|l}
m & \mathbf{A} \to \mathbf{B} & \\
n & \mathbf{A} & \\
 & \mathbf{B} & \to\text{E } m,\ n
\end{array}
$$

This rule is also known as MODUS PONENS.

Both rules can show that $\{A \to B, B \to C\} \vdash A \to C$:

$$
\begin{array}{l|l|l}
1 & A \to B & \\
2 & B \to C & \\
3 & \boxed{A} & \\
4 & \boxed{B} & \to\text{E } 1,\ 3 \\
5 & \boxed{C} & \to\text{E } 2,\ 4 \\
6 & A \to C & \to\text{I } 3\text{–}5
\end{array}
$$

4.2.1.6 Biconditional

↔I derives a biconditional between two sentences from the two subderivations deriving each from the other:

$$
\begin{array}{l|l|l}
i & \boxed{\mathbf{A}} & \\
j & \boxed{\mathbf{B}} & \\
k & \boxed{\mathbf{B}} & \\
l & \boxed{\mathbf{A}} & \\
 & \mathbf{A} \leftrightarrow \mathbf{B} & \leftrightarrow\text{I } i\text{–}j,\ k\text{–}l
\end{array}
$$

↔E derives either side of a biconditional from the biconditional and the other side:

$$\begin{array}{l|ll} m & \mathbf{A} \leftrightarrow \mathbf{B} & \\ n & \mathbf{A} & \\ & \mathbf{B} & \leftrightarrow\text{E } m, n \end{array} \qquad \begin{array}{l|ll} m & \mathbf{A} \leftrightarrow \mathbf{B} & \\ n & \mathbf{B} & \\ & \mathbf{A} & \leftrightarrow\text{E } m, n \end{array}$$

Both rules can show that $\{P \leftrightarrow Q\} \vdash Q \leftrightarrow P$:

$$\begin{array}{ll|ll} 1 & P \leftrightarrow Q & & \\ \hline 2 & \quad Q & & \\ \hline 3 & \quad P & \leftrightarrow\text{E } 1, 2 & \\ 4 & \quad P & & \\ \hline 5 & \quad Q & \leftrightarrow\text{E } 1, 4 & \\ \hline 6 & Q \leftrightarrow P & \leftrightarrow\text{I } 2\text{–}3, 4\text{–}5 & \end{array}$$

4.2.2 Shortcut Rules

Derivational rules can be chained to define **shortcut rules**.

4.2.2.1 Reiteration

A simple shortcut rule is REITERATION (**R**):

$$\begin{array}{l|ll} m & \mathbf{A} & \\ & \mathbf{A} & \text{R } m \end{array}$$

This rule abbreviates the following steps:

$$\begin{array}{l|ll} i & \mathbf{A} & \\ j & \mathbf{A} \wedge \mathbf{A} & \wedge\text{I } i, i \\ k & \mathbf{A} & \wedge\text{E } j \end{array}$$

4.2.2.2 Disjunctive Syllogism

The shortcut rule DISJUNCTIVE SYLLOGISM (**DS**) is:

m	$\mathbf{A} \vee \mathbf{B}$	
n	$\neg\mathbf{A}$	
	$\mathbf{B}$	DS m, n

m	$\mathbf{A} \vee \mathbf{B}$	
n	$\neg\mathbf{B}$	
	$\mathbf{A}$	DS m, n

This rule abbreviates the following steps:

i	$\mathbf{A} \vee \mathbf{B}$	
j	$\neg\mathbf{A}$	
k	$\mathbf{A}$	
l	$\mathbf{B}$	$\bot$E j, k
m	$\mathbf{B}$	
n	$\mathbf{B}$	R m
o	$\mathbf{B}$	$\vee$E i, k–l, m–n

4.2.2.3 Modus Tollens

The shortcut rule MODUS TOLLENS (**MT**) is:

m	$\mathbf{A} \to \mathbf{B}$	
n	$\neg\mathbf{B}$	
	$\neg\mathbf{A}$	MT m, n

This rule abbreviates the following steps:

i	$\mathbf{A} \to \mathbf{B}$	
j	$\neg\mathbf{B}$	
k	$\mathbf{A}$	
l	$\mathbf{B}$	$\to$E i, k
m	$\bot$	$\bot$I l, j
n	$\neg\mathbf{A}$	$\neg$I k–m

4.2.2.4 Double Negation Elimination

The shortcut rule DOUBLE NEGATION ELIMINATION (**DNE**) is:

$$\begin{array}{l|ll} m & \neg\neg\mathbf{A} & \\ & \mathbf{A} & \text{DNE } m \end{array}$$

This rule abbreviates the following steps:

$$\begin{array}{l|ll} i & \neg\neg\mathbf{A} & \\ j & \quad\neg\mathbf{A} & \\ k & \quad\bot & \bot\text{I } i, j \\ l & \mathbf{A} & \neg\text{E } j\text{–}k \end{array}$$

4.2.2.5 De Morgan's Laws

The famous DE MORGAN'S LAWS (**DeM**) are a collection of rules that govern interactions between $\neg$, $\wedge$, and $\vee$:

$$\begin{array}{l|ll} m & \neg(\mathbf{A} \wedge \mathbf{B}) & \\ & \neg\mathbf{A} \vee \neg\mathbf{B} & \text{DeM } m \end{array} \qquad \begin{array}{l|ll} m & \neg(\mathbf{A} \vee \mathbf{B}) & \\ & \neg\mathbf{A} \wedge \neg\mathbf{B} & \text{DeM } m \end{array}$$

$$\begin{array}{l|ll} m & \neg\mathbf{A} \vee \neg\mathbf{B} & \\ & \neg(\mathbf{A} \wedge \mathbf{B}) & \text{DeM } m \end{array} \qquad \begin{array}{l|ll} m & \neg\mathbf{A} \wedge \neg\mathbf{B} & \\ & \neg(\mathbf{A} \vee \mathbf{B}) & \text{DeM } m \end{array}$$

4.3 Derivational Concepts

We may define some important concepts in terms of derivability.

Theorems are derivable from *no* premises:

> **A** is a THEOREM iff $\emptyset \vdash \mathbf{A}$ (or: $\vdash \mathbf{A}$)

Sentences derivable from each other are **interderivable**:

> **A**,**B** are INTERDERIVABLE iff $\mathbf{A} \vdash \mathbf{B}$ and $\mathbf{B} \vdash \mathbf{A}$.

An **inconsistent** set permits the derivation of $\bot$:

> Γ is INCONSISTENT iff $\Gamma \vdash \bot$; else Γ is CONSISTENT.

And derivability is **monotonic** in that if a sentence is derivable from a set, then it is derivable from any expansion of that set:

MONOTONICITY. If $\Gamma \vdash \mathbf{C}$, then $\mathbf{A}, \Gamma \vdash \mathbf{C}$ (for any $\mathbf{A}$).

Monotonicity implies that theorems are derivable from every set whatsoever.

4.3.1 Soundness and Completeness

SOUNDNESS and COMPLETENESS imply that the semantic concepts (3) have *exact* derivational counterparts. By SOUNDNESS, if **A** is a theorem, then **A** is a validity. And, by COMPLETENESS, if **A** is a validity, then **A** is a theorem. For SL, theorems and validities coincide. This extends to other concepts too. For example, two sentences are equivalent just when they are interderivable. And, for another example, a set of sentences is unsatisfiable just when it is inconsistent.

Together, SOUNDNESS and COMPLETENESS allow us to interchange semantic and derivational methods. This is useful because each has its own compensatory costs and benefits. Recall:

$$A \therefore \neg A \vee (\neg A \rightarrow (A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow A_4))))$$

By SOUNDNESS, the premises entail the conclusion if the conclusion is derivable from them. So, we can show that they do by producing such a derivation. Here's one:

1	A	
2	$\quad \neg A$	
3	$\quad\quad \neg(A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow A_4)))$	
4	$\quad\quad \bot$	$\bot$I 1, 2
5	$\quad A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow A_4))$	$\neg$E 3–4
6	$\neg A \rightarrow (A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow A_4)))$	$\rightarrow$I 2–5
7	$\neg A \vee (\neg A \rightarrow (A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow A_4))))$	$\vee$I 6

Deriving this relies on different skills than calculating a semantic table. But those skills can be less tedious and more adaptable. Indeed, the same strategy can be used to show that $A \vdash \neg A \vee (\neg A \rightarrow (A_1 \rightarrow (A_2 \rightarrow \ldots (A_{98} \rightarrow A_{99})\ldots)$ in just 7 steps. This is nicer than calculating a semantic table with 2^{100} rows.

CHAPTER 5
Applications

This chapter focuses on applications of SL to natural language.

Remarkably, SL can be applied to *any* content fitting sentential form. The most spectacular application is to devices, like cellphones and laptops. These have transistors that treat electrical current as contents structured by sentential forms and perform logical operations on them. There'd be no digital age without sentential logic.

There are other applications. In linguistics, it is to the forms of languages. In mathematics, it is to the forms of theorems and proofs. In philosophy, it is to the forms of reality, knowledge, and morality.

None of these fields are done in SL. We ask philosophical questions ("Is there free will?") in our native languages. Mathematics has its own notation. And linguistics has been developing its own. How can we apply SL to subjects like these that do not "speak" SL?

We can by **rendering** sentences into SL. This involves selective attention: focusing on some aspects while abstracting away from others. There's no insinuation that what we abstract away is worse (or better). It is just that SL is not designed to capture content but *form*. Some sentences have richer forms than SL can capture. But this is no more a failure for SL than it is for a hammer not to turn a screw. A good toolbox has tools well-suited to each.

5.1 Truth-functional Semantics

SL is a good tool for representing **truth-functional** form. This concerns how a complex sentence's truth or falsity is determined by the truth or falsity of its subsentences. Consider:

Seattle is rainy and London is rainy.

Whether it is true or false depends on whether its subsentences 'Seattle is rainy' and 'London is rainy' are true or false. If both are true,

'Seattle is rainy and London is rainy' is true too. This illustrates one way (a *conjunctive* way) the truth or falsity of a sentence is determined by the truth or falsity of its parts. Truth-functional form concerns this sort of compositional determination in general.

SL represents truth-functional form if we interpret semantic values as **truth-values**: 1 as T (**true**) and 0 as F (**false**). This affects our *understanding* of a valuation, but not how semantic values are *determined*. So, we may rewrite semantic tables as **truth-tables**:

$\top$	$\top$
T	**T**

$\bot$	$\bot$
F	**F**

A	$\neg$**A**
T	**F**
F	**T**

A	**B**	**A** $\vee$ **B**
T	T	**T**
T	F	**T**
F	T	**T**
F	F	**F**

A	**B**	**A** $\wedge$ **B**
T	T	**T**
T	F	**F**
F	T	**F**
F	F	**F**

A	**B**	**A** $\rightarrow$ **B**
T	T	**T**
T	F	**F**
F	T	**T**
F	F	**T**

A	**B**	**A** $\leftrightarrow$ **B**
T	T	**T**
T	F	**F**
F	T	**F**
F	F	**T**

Truth-tables may be used to check whether semantic concepts apply.

5.1.1 Truth-functionality

Reinterpreting the semantics warrants renaming connectives **truth-functional**. This highlights their **truth-functionality**:

> A connective is TRUTH-FUNCTIONAL iff the truth-value of a sentence with it as its main connective is determined by the truth-value(s) of its subsentence(s).

Not all connectives are truth-functional. In English, 'because' is a connective. For example, putting 'because' between 'Ravens can fly' and 'Ravens have wings' yields:

Ravens can fly because ravens have wings.

This is true. Even so, 'because' is not truth-functional. Swap 'Ravens have wings' for another true sentence, such as 'Turtles have shells':

Ravens can fly because turtles have shells.

This is true. But it is false that ravens can fly *because* turtles have shells. Were 'because' truth-functional, swapping subsentences with the same truth-value would not change the truth-value of the sentence containing them. But it can. The truth-values of the parts of a 'because' statement *do not* determine the truth-value of the whole. So, 'because' is not truth-functional. While many English connectives are *not* truth-functional, all SL connectives are.

5.1.2 Truth-functional Concepts

Reinterpreting the semantics warrants renaming semantic concepts **truth-functional**. Recall:

Γ ENTAILS $\mathbf{C}$ iff there is no v such that $\min(\llbracket\Gamma\rrbracket_v) > \llbracket\mathbf{C}\rrbracket_v$

The new interpretation is: a set of premises entails a conclusion just when there is no valuation in which the premises are true and the conclusion is false. The other concepts are defined as before:

$\mathbf{A}$ is a VALIDITY (or TAUTOLOGY) iff $\emptyset \vDash \mathbf{A}$ (or: $\vDash \mathbf{A}$).

A validity (tautology) is true in all valuations.

$\mathbf{A}$ is a CONTRADICTION iff $\mathbf{A} \vDash \bot$.

A contradiction is false in all valuations.

$\mathbf{A}$ is a CONTINGENCY iff neither $\vDash \mathbf{A}$ nor $\mathbf{A} \vDash \bot$.

A contingency is true in some valuations and false in others.

$\mathbf{A}, \mathbf{B}$ are EQUIVALENT iff $\mathbf{A} \vDash \mathbf{B}$ and $\mathbf{B} \vDash \mathbf{A}$.

Sentences are equivalent when their truth-values never differ.

Γ is SATISFIABLE iff $\Gamma \nvDash \bot$.

A set of sentences is satisfiable when a valuation makes them all true.

The semantic tests for these concepts remain unchanged.

5.1.3 Derivations

Reinterpreting the semantics does not affect derivations.

5.2 Renderings

We apply SL to natural languages by **rendering** their sentences into SL. We focus on sentences fit for rendering: **declarative sentences** (5.2.1). Some English connective phrases ('not', 'or', 'and', 'if', 'if and only if') may be rendered as truth-functional connectives ($\neg, \vee, \wedge, \rightarrow, \leftrightarrow$). English sentences without these connective phrases usually get rendered as atomic (5.2.2), while those with them usually get rendered as complex (5.2.3). A rendering is temporary: an SL letter assigned a sentence at one time may be assigned a different sentence at another time. And there are many complexities and nuances that arise. Even so, rendering is powerful: it allows us to apply SL to study the truth-functional forms of English sentences.

5.2.1 Declarative Sentences

Reinterpreting the semantics limits rendering: only to sentences apt for having a truth-value. Such a sentence *states* something: it *declares*–truly or falsely–how things are. It is a **declarative sentence**:

A sentence is DECLARATIVE iff it is apt for having a truth-value.

There are many reasons why a sentence may be inapt for having a truth-value. Some are just ungrammatical gibberish:

Dog cat ate the the.

Others are inapt for being (apparently) grammatical gibberish:

The foo bleened the glug.

Gibberish, grammatical or not, fails to *state* anything.

Non-gibberish, grammatical sentences may be nondeclarative. **Expressive** sentences do not state but *express* or *emote*:

Grrrrrrr! Ouch! Oops!

Expressing or emoting may truly or falsely represent how one feels. But the *expression* or *emotion* is truth-inapt.

Imperative sentences do not state but *command*:

Shut the door! Go away! Stop!

It may be true or false that a command was obeyed. But the *command* is truth-inapt.

Interrogative sentences do not state but *ask*:

How's Annie? What year is it? Who can it be now?

An answer may be true or false. But the *question* is truth-inapt.

Whether a sentence is *apt* for having a truth-value is not the same as whether we *know* its truth-value. Consider:

Andromeda Galaxy now has an even number of stars.

This is true or false, although we'll never know which.

5.2.2 Atomic Sentences

English sentences without words like 'not', 'or', 'and', 'if', 'if and only if' are often not truth-functionally complex. So, they should be rendered as atomic sentences. To illustrate, consider how to render 'Fido barks'. Both $\top$ and $\bot$ are atomic. Suppose we render it as:

$$\top \quad \Longleftarrow \quad \text{Fido barks.}$$

This renders 'Fido barks' as a tautology: true in all valuations. But it could be false (Fido needn't bark). Or suppose we choose:

$$\bot \quad \Longleftarrow \quad \text{Fido barks.}$$

This renders 'Fido barks' as a contradiction: false in all valuations. But it could be true (Fido might bark). Instead, 'Fido barks' is a contingency: it is neither a tautology nor a contradiction. So, it should not be rendered as $\top$ or $\bot$ but as a letter:

$$A \quad \Longleftarrow \quad \text{Fido barks.}$$

Any other letter could have been used, provided it was not already in

use. Suppose it was already in use, as in:

$$\begin{array}{rcl} A & \Longleftarrow & \text{13 is prime.} \\ A & \Longleftarrow & \text{Fido barks.} \end{array}$$

What is rendered as A: something about a number or a dog? The rendering is ambiguous. We avoid ambiguity by not rendering multiple sentences as one letter.

We should also avoid rendering the same sentence as multiple letters. Consider:

$$\begin{array}{rcl} A & \Longleftarrow & \text{Fido barks.} \\ B & \Longleftarrow & \text{Fido barks.} \end{array}$$

Our semantics assigns truth-values to letters *independently*. So, there are valuations that assign A, B *different* truth-values. But then this rendering interprets 'Fido barks' as being both true and false in some valuations. That conflicts with truth and falsity being exclusive. So, we should avoid rendering one sentence as multiple letters.

The best policy is to render a sentence *uniquely*. If a sentence is rendered at all, it is by exactly one letter.

We must also avoid rendering complex sentences with truth-functional form as letters without truth-functional form. Suppose we also rendered 'Fido does not bark' as B:

$$B \quad \Longleftarrow \quad \text{Fido does not bark.}$$

The sentences 'Fido does not bark' and 'Fido barks' are opposed. But our renderings obscure how. They are opposed like 'Gold glitters' opposes 'Gold does not glitter'. The opposition is *formal*: the same as the opposition between any sentence and its negation. A better rendering would display this. This is done with $\neg$:

$$\begin{array}{rcl} A & \Longleftarrow & \text{Fido barks.} \\ \neg A & \Longleftarrow & \text{Fido does not bark.} \\ P & \Longleftarrow & \text{Gold glitters.} \\ \neg P & \Longleftarrow & \text{Gold does not glitter.} \end{array}$$

In general, we should render English connectives as truth-functional connectives. Not doing so squanders the power of using SL. To illustrate, consider this rendering:

$$\begin{array}{rcl} Q & \Longleftarrow & \text{It's windy and rainy.} \\ W & \Longleftarrow & \text{It's windy.} \\ R & \Longleftarrow & \text{It's rainy.} \end{array}$$

Had we rendered 'It's windy and rainy' as the *conjunction* $W \wedge R$, we'd have captured how 'It's windy and rainy' is true only if each of conjuncts 'It's windy' and 'It's rainy' are true. But this cannot be done by rendering 'It's windy and rainy' as a letter.

Renderings should display as much form as SL allows.

5.2.3 Complex Sentences

Words like 'not', 'or', 'and', 'if', 'if and only if' may often be rendered as $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$. But their semantics diverges from the meanings of these English words. Whereas SL was *designed* to display form, English *evolved* messily for other uses. Divergences are expected.

5.2.3.1 Negation

We read $\neg$ as 'not'. Other English words also can express negation. Sometimes the prefix 'un' does. For example, 'unashamed' means *not ashamed*. Nothing is ashamed and unashamed, and nothing is neither. So, this rendering is fine:

$$\begin{array}{rcl} A & \Longleftarrow & \text{Joe is ashamed.} \\ \neg A & \Longleftarrow & \text{Joe is unashamed.} \end{array}$$

But 'un' does not always express negation. For example, 'unhappy' does *not* mean *not happy*. Perhaps Joe is placid: neither happy nor unhappy. The following rendering omits this:

$$\begin{array}{rcl} H & \Longleftarrow & \text{Joe is happy.} \\ \neg H & \Longleftarrow & \text{Joe is unhappy.} \end{array}$$

Neither H nor $\neg H$ alone express Joe's placidness. And $H \wedge \neg H$ distorts his placidness as a contradiction. A better rendering is:

$$\begin{array}{rcl} H & \Longleftarrow & \text{Joe is happy.} \\ U & \Longleftarrow & \text{Joe is unhappy.} \end{array}$$

Now we express Joe's placidness as $\neg H \wedge \neg U$ or, equivalently by De Morgan's Laws (4.2.2.5) and SOUNDNESS, as $\neg(H \vee U)$).

Negative expressions (like 'un') aren't always best rendered as $\neg$. It is not always straighforward which English expressions are best rendered as truth-functional connectives.

This is also illustrated by the sentence: 'Kai does not like soup, he loves it'. Suppose we render this as:

$$\begin{array}{rcl} L & \Longleftarrow & \text{Kai likes soup.} \\ \neg L & \Longleftarrow & \text{Kai does not like soup.} \\ M & \Longleftarrow & \text{Kai loves soup.} \\ \neg L \wedge M & \Longleftarrow & \text{Kai does not like soup, he loves it.} \end{array}$$

In this context, loving is a degree of liking. The original sentence does *not* deny that Kai likes soup. Instead, it emphasizes that he does not *merely* like it but that he likes it *a lot*. This shows why the rendering is bad. $\neg L \wedge M$ is true only if Kai loves soup without liking it. But what we meant to say is true only if Kai likes soup *a lot*. This suggests that M is a better rendering of the original sentence. But even it is not ideal. Kai loves soup only if he likes it. So, M should entail L. But, as the truth-table method could verify, it is not. This can be fixed by rendering the original sentence as:

$$L \wedge M \quad \Longleftarrow \quad \text{Kai likes soup and loves soup.}$$

This rules out Kai's loving without liking soup. Its explicitness sounds clunky. But a good rendering needn't sound elegant.

5.2.3.2 Disjunctions

We read $\vee$ as 'or' in the *inclusive* sense that the disjunction is true if *one or both* disjuncts are true. This contrasts with an *exclusive* sense on which the disjunction is true when *one* disjunct is true. This is the sense when offered soup or salad at the deli (you must choose *one*). An exlusive 'or' is rendered by combining connectives:

$$\begin{array}{rcl} P & \Longleftarrow & \text{Ali got soup.} \\ D & \Longleftarrow & \text{Ali got salad.} \\ P \vee D & \Longleftarrow & \text{Ali got soup [}\textit{inclusive}\text{] or salad} \\ (P \vee D) \wedge \neg(P \wedge D) & \Longleftarrow & \text{Ali got soup [}\textit{exclusive}\text{] or salad} \end{array}$$

Context usually guides us on which sense to render 'or' as.

Sentences with 'unless' may be rendered as disjunctions in either sense. To illustrate an inclusive rendering, consider these:

Ori will fail the exam unless he studies.
Unless Ori studies, Ori will fail the exam.

These are stylistic variants. They should be rendered the same. Each is false only if Ori passes the exam without studying. But then each is true if Ori fails or studies. So, our rendering is:

$$\begin{array}{rcl} F & \Longleftarrow & \text{Ori will fail the exam.} \\ S & \Longleftarrow & \text{Ori will study.} \\ F \vee S & \Longleftarrow & \text{Ori will fail the exam unless he studies.} \end{array}$$

Because the order of disjuncts makes no difference to the truth-value of a disjunction, we could have rendered each as $F \vee S$. But it is customary to preserve the order of the sentence rendered.

We may illustrate an exclusive rendering of 'unless' with:

Ty will vote unless Mai runs.

This is false either if Ty doesn't vote and Mai doesn't run or if Mai runs and Ty votes. So, we may render it as:

$$\begin{array}{rcl} V & \Longleftarrow & \text{Ty votes.} \\ R & \Longleftarrow & \text{Mai runs.} \\ (V \vee R) \wedge \neg(V \wedge R) & \Longleftarrow & \text{Ty will vote unless Mai runs.} \end{array}$$

This rendering is equivalent to the biconditional $V \leftrightarrow \neg R$.

5.2.3.3 Conjunctions

We read $\wedge$ as 'and' in an *atemporal* sense that ignores the temporal order of the conjuncts. Consider:

$$\begin{array}{rcl} L & \Longleftarrow & \text{Socrates lived.} \\ D & \Longleftarrow & \text{Socrates died.} \end{array}$$

The semantics for $\wedge$ ensure that $L \wedge D$ and $D \wedge L$ are both true if D and L are true. This conflicts with how we often use 'and' in a *temporal* sense on which the conjunction is true only if the left conjunct occurs

before the right. This explains why 'Socrates lived and died' seems true (he lived *before* he died) but 'Socrates died and lived' may not. We cannot render the temporal 'and' in SL because *no* connective captures *temporal* form. We can only use letters:

$$\begin{array}{rcl} A & \Longleftarrow & \text{Socrates lived before dying.} \\ B & \Longleftarrow & \text{Socrates died before living.} \end{array}$$

But, presumably, 'Socrates lived before dying' entails 'Socrates lived'. Our renderings miss this because $A \nvDash L$. A better rendering is:

$$\begin{array}{rcl} (L \wedge D) \wedge A & \Longleftarrow & \text{Socrates lived and [}\textit{then}\text{] died.} \\ (L \wedge D) \wedge B & \Longleftarrow & \text{Socrates died and [}\textit{then}\text{] lived.} \end{array}$$

Because $(L \wedge D) \wedge A \vDash L$, this rendering captures 'Socrates lived' as a consequence of 'Socrates lived and [*then*] died'.

English can *compress* conjuncts in ways SL can't. Consider:

Ann is kind and sly.

This rendering is bad because 'sly' is not a declarative sentence:

$$\begin{array}{rcl} K & \Longleftarrow & \text{Ann is kind} \\ S & \Longleftarrow & \text{sly} \end{array}$$

A better rendering *decompresses* the conjuncts:

$$\begin{array}{rcl} K & \Longleftarrow & \text{Ann is kind.} \\ S & \Longleftarrow & \text{Ann is sly.} \\ K \wedge S & \Longleftarrow & \text{Ann is kind and sly.} \end{array}$$

English allows nuances that SL ignores. To illustrate, consider:

Ann is kind and sly.
Ann is kind but sly.
Although Ann is sly, she is kind.

Because SL ignores the nuances, these must be rendered alike. The order of conjuncts does not affect the conjunction's truth-value. So, it can be $K \wedge S$ or $S \wedge K$. But it is ideal to preserve the original order.

5.2.3.4 Conditionals

We read $\rightarrow$ as 'if...then...' in the sense fixed by $\rightarrow$'s truth-table: the **material conditional**. Consider:

If Alf is a human, then Alf is a mammal.

This may be rendered as:

$$\begin{array}{rcl} H & \Longleftarrow & \text{Alf is a human.} \\ M & \Longleftarrow & \text{Alf is a mammal.} \\ H \rightarrow M & \Longleftarrow & \text{If Alf is a human, then Alf is a mammal.} \end{array}$$

Other ways of expressing the same material conditional include:

Alf is a human only if Alf is a mammal.
Alf is a mammal if Alf is a human.

However expressed, material conditionals capture **necessary conditions** and **sufficient conditions**. If $\mathbf{A} \rightarrow \mathbf{C}$ is true, the truth of $\mathbf{A}$ *suffices*, or is a *sufficient condition*, for the truth of $\mathbf{C}$ and the truth of $\mathbf{C}$ is *necessary*, or is a *necessary condition*, for the truth of $\mathbf{C}$.

The material conditional differs from other senses expressible by 'if...then...' and its variations. One of these is **hypothetical**:

If he had left earlier, then he would not be late.

Another sense is **causal**, as in reading the next sentence as saying that Sue's throwing *caused* the window's breaking:

If Sue threw the rock, then she broke the window.

Yet another sense is **implication**, as in reading the next sentence as saying that Alf's being human *implies* his being a mammal:

If Alf is a human, then Alf is a mammal.

Senses like these often convey some *relevant* connection between the antecedent and consequent. When the antecedent and consequent are irrelevant, the conditional may then seem false:

If Sue threw the rock, then dogs bark.

But the material conditional $\rightarrow$ does *not* require relevance. Consider:

$$\begin{array}{rcl} R & \Longleftarrow & \text{Sue threw the rock.} \\ B & \Longleftarrow & \text{Dogs bark.} \\ R \to B & \Longleftarrow & \text{If Sue threw the rock, then dogs bark.} \end{array}$$

The truth-table shows that $R \to B$ is true if B is true. Their irrelevance is irrelevant to the semantics of the material conditional.

Another difference is that an implication is "directional" in a way that a material conditional needn't be. Suppose Alf is a human mammal: H and M are true. By $\to$'s truth-table, $H \to M$ is true. But so is its **converse**: $M \to H$. Reading $\to$ as implication, we get that Alf's being human implies his being a mammal *and* his being a mammal implies his being a human. But only the first implication is correct. In general, $\to$ does not mean implication.

5.2.3.5 Biconditionals

We read $\leftrightarrow$ as 'if and only if' in the sense fixed by $\leftrightarrow$'s truth-table: the **material biconditional**. Consider:

Al sleeps if and only if Al snores.

This may be rendered as:

$$\begin{array}{rcl} L & \Longleftarrow & \text{Al sleeps.} \\ N & \Longleftarrow & \text{Al snores.} \\ L \leftrightarrow N & \Longleftarrow & \text{Al sleeps if and only if Al snores.} \end{array}$$

This is equivalent to conjoining a conditional and its converse:

$$(L \to N) \wedge (N \to L) \quad \Longleftarrow \quad \text{Al sleeps if and only if Al snores.}$$

So, a material biconditional **A** $\leftrightarrow$ **B** captures *both* **necessary and sufficient conditions**: **A** is *both* necessary *and* sufficient for **B**.

5.2.3.6 Ambiguities

By design, SL's syntax determines a connective's scope. Ambiguity is impossible. But English is full of ambiguities. Consider:

It's not the case that it's raining and it's windy.

This is ambiguous. One reading says that it is not both rainy and windy. Another says that it is not rainy but it is windy. The ambiguity means that we have *two* options for rendering:

$$\begin{array}{rcl} \neg(R \wedge W) & \Longleftarrow & \text{It is not both rainy and windy.} \\ \neg R \wedge W & \Longleftarrow & \text{It is not rainy and it is windy.} \end{array}$$

Context often hints at how to disambiguate. But not always.

5.2.4 Truth

Form alone rarely determines whether a sentence is true. It does not determine whether Tim lied on his taxes. Rendering confirms this:

$$T \quad \Longleftarrow \quad \text{Tim lied on his taxes.}$$

The truth-table for T shows that it is a contingency:

T	T
T	**T**
F	**F**

In general, form does not determine the truth-value of any letter. That's a matter of *content*. Letters are contingencies.

But form can determine whether some complex sentences are true or false. A tautology is true in virtue of its truth-functional form. For example, $T \vee \neg T$ is a tautology:

T	$T \vee \neg T$
T	T **T** F T
F	F **T** T F

By contrast, a contradiction is false in virtue of its truth-functional form. For example, $T \wedge \neg T$ is a contradiction:

T	$T \wedge \neg T$
T	T **F** F T
F	F **F** T F

Not all complex sentences are tautologies or contradictions. Some are contingencies. Form alone does not determine their truth-values. But it does for tautologies and contradictions.

These points help explain why some bad renderings are bad. Consider, again, the earlier rendering (5.2.3):

$$\begin{array}{rcl} H & \Longleftarrow & \text{Joe is happy.} \\ \neg H & \Longleftarrow & \text{Joe is unhappy.} \end{array}$$

This suggests rendering 'Joe is happy or unhappy' as $H \vee \neg H$. Because $H \vee \neg H$ is a tautology, this renders 'Joe is happy or unhappy' as a tautology. But it may not even be true. Joe might be placid, so neither happy nor unhappy. The rendering incorrectly renders 'Joe is happy or unhappy' as a tautology when it is not.

5.2.5 Equivalence

Standards shift for when sentences are "equivalent". 'Cy is always late' is *practically equivalent* to 'Cy is never on time' if our goal is to say why Cy missed class. But 'To be or not to be, that is the question' is not *literarily equivalent* to 'I wonder: should I live or die?'. The standard in SL is *truth-functional equivalence*: sentences are equivalent when their forms ensure that their truth-values cannot differ. Recall:

$$\begin{array}{rcl} L & \Longleftarrow & \text{Socrates lived.} \\ D & \Longleftarrow & \text{Socrates died.} \end{array}$$

When rendered into SL, this pair is equivalent:

L	D	$L \wedge D$	$D \wedge L$
T	T	T **T** T	T **T** T
T	F	T **F** F	F **F** T
F	T	F **F** T	T **F** F
F	F	F **F** T	F **F** F

More vividly, *any* pair of tautologies is equivalent. To illustrate:

A	B	$A \rightarrow A$	$B \vee \neg B$
T	T	T **T** T	T **T** F F
T	F	T **T** T	F **T** T F
F	T	F **T** F	T **T** F T
F	F	F **T** F	F **T** T F

Similarly, *any* pair of contradictions is equivalent. The negations of the previous pair $\neg(A \rightarrow A), \neg(B \vee \neg B)$ illustrate this.

This also illustrates that *truth-functional* equivalence is not the same as equivalence in *meaning*.

5.2.6 Consistency

Some sentences cannot be jointly true. Consider these two:

Charles III is the British king. Biden is the British king.

These cannot be jointly true *given how the British monarchy works*. But this does not show that they are *inconsistent* (4.3), or *unsatisfiable* (3.4.6), in our defined senses. That depends just on their *form*. And rendering them reveals them to be *satisfiable*:

$$\begin{array}{ll} C & \Longleftarrow \text{ Charles III is the British king.} \\ B & \Longleftarrow \text{ Biden is the British king.} \end{array}$$

Any two letters are satisfiable, including these:

C	B	C	B
T	T	**T**	**T**
T	F	T	F
F	T	F	T
F	F	F	F

So, the original sentences cannot jointly be true but *not* because of their truth-functional form. But truth-functional form alone can make some sentences unsatisfiable. Consider the set $\{C \rightarrow B, \neg B, C\}$:

B	C	$C \rightarrow B$	$\neg B$	C
T	T	T T T	F T	T
T	F	F T T	F T	F
T	T	T F F	T F	T
F	F	F T F	T F	F

The truth-table shows that this set is unsatisfiable.

In English, 'consistent' is often used to mean something closer to our defined sense of *satisfiable* (3.4.6) than to our defined sense of *consistent* (4.3). Given SOUNDNESS and COMPLETENESS, we may use 'consistent' and 'satisfiable' interchangeably when applying those defined senses to English.

5.2.7 Reasoning

Reasoning involves giving *reasons* or *justifications* for a conclusion. We construe this as an argument. Recall, an argument is not a quarrel but a sequence $\Gamma \therefore \mathbf{C}$ of premises Γ and conclusion $\mathbf{C}$. The premises and conclusion must be declarative sentences. Nothing else limits what they may be. The premises may include the conclusion, or be empty. This allows flexibility in applying our notion of an argument. We apply it to premises *offered as reasons* for a conclusion. To illustrate, we may wonder whether Tim is a crook. We may demand a reason–an *argument*–to believe that he is. Suppose we have one:

> If Tim lied on his taxes, then Tim is a crook.
> Tim lied on his taxes.
> $\therefore$ Tim is a crook.

This captures one line of reasoning. But is it a *good* argument?

5.2.8 Valid Arguments

One way to evaluate an argument is whether its conclusion *follows from* its premises: whether it is *impossible* for the premises to be true and the conclusion false. If so, the argument is **valid**:

> An argument is VALID iff it is impossible for the premises to be true and the conclusion false; else it is INVALID.

Validity in this sense applies to *arguments*, not *sentences*. So, it is *not* the notion defined before (3.4.2). The label 'valid' does double-duty for both notions. While this terminology may be confusing, it is entrenched. Context should clarify which notion is intended.

Validity appeals to a notion of *possibility*. There are many, and most are not well-understood. But we may sharpen ours by appeal to entailment. When the premises of an argument entail its conclusion, their truth-functional forms alone make it impossible for the premises to be true and the conclusion false. Entailment implies validity:

$$\text{If } \Gamma \vDash \mathbf{C}, \text{ then } \Gamma \therefore \mathbf{C} \text{ is valid.}$$

Truth-functional form won't capture all the valid arguments there are (III-IV). But our focus now is on arguments that are valid *because* of truth-functional form. To illustrate, consider:

$$\begin{array}{ll} T & \Longleftarrow \text{ Tim lied on his taxes.} \\ C & \Longleftarrow \text{ Tim is a crook.} \end{array}$$

We may then render the argument above $\{T \to C, T\} \therefore C$ as:

$$\begin{array}{ll} & T \to C \\ & T \\ \therefore & C \end{array}$$

Because SL is both sound and complete, the semantic and derivational methods coincide. So, we can use either method to show whether or not an argument is valid.

We may construct a truth-table to see whether or not the premises entail the conclusion:

T	C	T	$T \to C$	C
T	T	T	T T T	T
T	F	T	T F F	F
F	T	F	F T T	T
F	F	F	F T F	F

This shows that, indeed, $\{T \to C, T\} \vDash C$.

Or we may derive the conclusion from the premises:

1	$T \to C$	
2	T	
3	C	$\to$E 1, 2

This shows that, indeed, $\{T \to C, T\} \vdash C$.

None of this means Tim *is* a crook (though he may be). Entailment is a *formal* notion. Form may determine that the conclusion is true *if* the premises are true. But this usually implies nothing about whether the premises or conclusion *are* true or false. It is not a matter of *form* whether Tim lied on his taxes, or whether that makes him a crook. That's a matter of *content*.

Valid arguments play a crucial role in many areas of inquiry. In mathematics, a theorem may be proved by showing that the axioms entail it: that is, by producing a valid argument from the axioms to the theorem. In science, a hypothesis may be tested by the predictions it entails: that is, by producing a valid argument from the hypothesis (and background premises) to a prediction, and then observing

whether or nor the prediction holds.

5.2.9 Soundness

If the premises of a valid argument are true, then its validity ensures that the conclusion is true. Such an argument is **sound**:

> An argument is SOUND iff it is valid and its premises are true; else it is UNSOUND.

Soundness, in this sense, applies to *arguments*. It is not the *metalogical* result of SOUNDNESS (4.2). So, we label them differently.

There is a crucial difference between validity and soundness. Validity is a *modal* notion: it captures the *impossibility* of true premises and a false conclusion. That alone, however, does not generally imply whether the premises or conclusion *actually* are true or false. Soundness requires validity. But it goes beyond. Soundness also requires that the premises *actually* be true. That, together with the validity of a sound argument, requires that the conclusion also *actually* be true.

5.2.10 Quirks

Validity has *something* to do with good reasoning. But the relationship is quirky. Soundness inherits the quirks because it requires validity. Studying the quirks will sharpen just how validity and good reasoning are related, and how they're not.

A CIRCULAR ARGUMENT has its conclusion as a premise: $\{\ldots, \mathbf{C}, \ldots\} \therefore \mathbf{C}$. Circular arguments are bad. But not because they're invalid. They must be valid. This can be shown by semantic or derivational methods. By semantics: we show that no valuation makes the conclusion $\mathbf{C}$ false and the premises $\ldots, \mathbf{C}, \ldots$ true. No valuation can because it would require $\mathbf{C}$ being false and true. By derivations: we show that $\{\ldots, \mathbf{C}, \ldots\} \vdash \mathbf{C}$. The derivation takes one step: use rule R to reiterate $\mathbf{C}$. By SOUNDNESS, the argument is valid. Why is it still bad? Because of its intended application: to offer its premises as support for its conclusion. A circular argument offers its conclusion in support of itself. (As philosophers say, it **begs the question**.) That's bad.

Some arguments have **inconsistent premises**. They cannot be sound, but must be valid. This can be shown by semantic or derivational methods. By semantics: we show that no valuation makes the

conclusion **C** false and the premises Γ true. If Γ is inconsistent, no valuation makes Γ all true. But then no valuation makes Γ all true and **C** false. By derivations: we show that $\Gamma \vdash \mathbf{C}$. If Γ is inconsistent, then $\Gamma \vdash \bot$. But $\bot \vdash \mathbf{A}$ for any sentence **A**, including **C**. So, $\Gamma \vdash \mathbf{C}$. By SOUNDNESS, the argument is valid. Why is it still bad? As before, because of its intended application: to offer its premises as support for its conclusion. An argument with inconsistent premises offers false support for its conclusion. That's bad.

Some arguments have **tautological** conclusions. They are valid. This can be shown by semantic or derivational methods. By semantics: we show that no valuation makes the conclusion **C** false and the premises Γ true. If **C** is a tautology, every valuation makes it true. By derivations: we show that $\Gamma \vdash \mathbf{C}$. If **C** is a tautology ($\vDash \mathbf{C}$), then, by SOUNDNESS, it is a theorem ($\vdash \mathbf{C}$). By monotonicity and SOUNDNESS, if $\vdash \mathbf{C}$, then $\Gamma \vDash \mathbf{C}$. An argument with a tautological conclusion may be bad even though it is valid. Consider:

Pigs fly.
Tim is virtuous.
$\therefore$ Tim is a crook or it is not the case that Tim is a crook.

Using this rendering:

$$\begin{array}{lll} P & \Longleftarrow & \text{Pigs fly.} \\ V & \Longleftarrow & \text{Tim is virtuous.} \\ C & \Longleftarrow & \text{Tim is a crook.} \end{array}$$

we get:

P
V
$\therefore\ C \vee \neg C$

This is valid. Why is it still bad? Again, because of its intended application: to offer its premises as support for its conclusion. An argument with a tautological conclusion is valid *regardless* of its premises, whether irrelevant or false. That's bad.

Monotonicity means that adding a premise cannot produce an invalid argument from a valid one. But it can produce a bad argument from a good one. To illustrate, recall:

$$\{T \rightarrow C, T\} \therefore C$$

It's valid. We may suppose it's also sound. So, it's good. Now, add the

premise 'Pigs fly'. The new argument is:

$$\{T \rightarrow C, T, P\} \therefore C$$

By monotonicity, it is valid. But it's bad. The new argument is unsound: pigs don't fly. It's also bad for another reason: the added premise is irrelevant. Or, adding 'Tim is a crook' gives:

$$\{T \rightarrow C, T, C\} \therefore C$$

By monotonicity, it is valid. But it is bad because it is circular. Or, adding 'It is not the case that Tim lied on his taxes' gives:

$$\{T \rightarrow C, T, \neg T\} \therefore C$$

By monotonicity, it is valid. But it is bad because its premises are inconsistent. Monotonicity preserves validity, not goodness.

Adding a premise can produce a valid argument from an invalid argument. Suppose $\{\mathbf{A_1}, ..., \mathbf{A_n}\} \therefore \mathbf{C}$ is invalid. Conjoin the premises $\mathbf{A_1} \wedge ... \wedge \mathbf{A_n}$. Make the conjunction the antecedent of a conditional whose consequent is the conclusion: $\mathbf{A_1} \wedge ... \wedge \mathbf{A_n} \rightarrow \mathbf{C}$. The result is:

$$\{\mathbf{A_1}, ..., \mathbf{A_n}, \mathbf{A_1} \wedge ... \wedge \mathbf{A_n} \rightarrow \mathbf{C}\} \therefore \mathbf{C}$$

And this argument is valid. Replacing an invalid argument with a valid one in this way can be useful: it allows us to focus on determining whether it is bad in some other way.

5.3 Limitations

The only forms SL captures are those of the truth-functional connectives. These are impressive: they can implement computers. But they cannot capture *other* worthy forms.

One such form involves **quantities**. Consider the argument:

Cy voted.
$\therefore$ Someone voted.

Neither the premise nor the conclusion has any connectives. So, each must be rendered as an atomic sentence:

$$\begin{array}{rcl} C & \Longleftarrow & \text{Cy voted.} \\ S & \Longleftarrow & \text{Someone voted.} \end{array}$$

We may render this as $\{C\} \therefore S$. SL only accounts for valid arguments for which no valuation assigns the premises to be true and the conclusion false. But the truth-table shows there is such a valuation:

C	S	C	S
T	T	T	T
T	F	**T**	**F**
F	T	F	T
F	F	F	F

So, SL cannot explain why the argument is valid. Or consider:

Everyone voted.
$\therefore$ Someone voted.

Neither the premise nor the conclusion has any connectives. So, each must be rendered as an atomic sentence:

E $\Longleftarrow$ Everyone voted.
S $\Longleftarrow$ Someone voted.

The argument may be rendered as $\{E\} \therefore S$. Its truth-table is:

E	S	E	S
T	T	T	T
T	F	**T**	**F**
F	T	F	T
F	F	F	F

And so SL must regard the argument $\{E\} \therefore S$ as invalid. But what we wanted to render is clearly valid.

Nor is the problem confined to validity. It also arises for equivalence. These two sentences are equivalent:

Not everyone voted.
Someone did not vote.

The first sentence can be neatly rendered as:

$\neg E$ $\Longleftarrow$ Not everyone voted.

But how to render the second sentence? Not like this:

$\neg S$ $\Longleftarrow$ It is not the case that someone voted.

That says that *no one* voted. That is *not* equivalent to 'Not everyone voted'. It seems, instead, that we must render the second sentence as an atomic sentence:

$$Q \quad \Longleftarrow \quad \text{Someone did not vote.}$$

But Q is *inequivalent* to $\neg E$. The truth-table shows this:

E	Q	$\neg E$	Q
T	T	F T	T
T	F	F T	F
F	T	T F	T
F	F	T F	F

So, SL cannot render the two sentences as equivalent.

The problem also extends to consistency. These two sentences cannot both be true:

Everyone voted.
It is not the case that someone voted.

The most straightforward renderings of these into SL are:

$$\begin{array}{rcl} E & \Longleftarrow & \text{Everyone voted.} \\ \neg S & \Longleftarrow & \text{It is not the case that someone voted.} \end{array}$$

Is the set $\{E, \neg S\}$ consistent? The truth-table shows that it is:

E	S	E	$\neg S$
T	T	T	F T
T	F	T	T F
F	T	F	F T
F	F	F	T F

These sentences cannot be rendered in SL as inconsistent.

These limitations do not undermine how SL nicely captures *truth-functional* form. But they show that there *other forms* that SL does *not* capture. This motivates crafting a richer formal language that can.

PART III

Monadic Form

CHAPTER 6
Syntax

This chapter focuses on the syntax of a formal language, ML, capturing monadic form.

6.1 Expressions

ML's alphabet includes SL's connectives and punctuation but adds new kinds of simple expressions: **terms**, **predicates**, and **quantifiers**:

The ALPHABET OF ML consists in:

TERMS	**Names**	$a, b, \ldots, r, \ldots, a_8, \ldots, r_8, \ldots$
	Variables	$s, t, u, v, w, x, y, z, s_0, \ldots, z_0, \ldots$
PREDICATES	**0-place**	$A^0, B^0, \ldots, Z^0, A^0_0, B^0_0, \ldots, Z^0_0, \ldots$
		$\top, \bot$
	1-place	$A^1, B^1, \ldots, Z^1, A^1_0, B^1_0, \ldots, Z^1_0, \ldots$
QUANTIFIERS		$\exists, \forall$
CONNECTIVES		$\neg, \vee, \wedge, \rightarrow, \leftrightarrow$
PUNCTUATION		$(\ ,\ )$

Expressions are defined in terms of the simple expressions:

An ML EXPRESSION is any sequence from its alphabet.

But what are the new kinds of simple expressions?

6.1.1 Predicates

Predicates are expressions with "slots". Each has a fixed number: its *degree* or *places*. ML is *monadic* because restricts its predicates to a degree of no more than 1. Uppercase letters $A, \ldots, Z$ (subscripted as needed) are for predicates. Superscripts signal degree. So, F^1 is a 1-place predicate and Q^0_7 is a 0-place predicate.

6.1.2 Terms

Terms "fit" into a predicate's "slots". There are two kinds: **names** and **variables**. Lowercase letters (subscripted as needed) are used for both: a to r for names, s to z for variables. Their syntactic difference is that variables interact with *quantifiers* while names don't.

6.1.3 Quantifiers

Quantifiers $\forall, \exists$ are devices for *binding* variables. Binding a variable provides a syntactic marker for how to interpret the semantics for expressions with variables. For now, our focus is just on syntax.

A quantifier formula has the form $\exists \boldsymbol{v}\mathbf{A}$ or $\forall \boldsymbol{v}\mathbf{A}$. We may be more specific: a formula that contains $\exists \boldsymbol{v}$ or $\forall \boldsymbol{v}$ is a $\boldsymbol{v}$**-quantifier formula**. A formula is $\boldsymbol{v}$**-quantifier-free** if it contains neither. And a formula is **quantifier-free** if it is $\boldsymbol{v}$-quantifier-free for all $\boldsymbol{v}$.

6.2 Formulas

Unlike SL, ML's atomic sentences have structure: they are built from predicates, terms, and quantifiers. Specifying *how* relies on the intermediate notion of a **formula**. We first define formulas and then sentences in terms of them. We use new metavariables: $\boldsymbol{P}$,$\boldsymbol{Q}$,... for predicates, $\boldsymbol{s}$,$\boldsymbol{t}$,... for terms.

An ML ATOMIC FORMULA is generated by these rules:

1. $\top$ and $\bot$ are atomic formulas.
2. Every $\ulcorner \boldsymbol{P}^k(\mathbf{s}_k) \urcorner$ is an atomic formula (where $\mathbf{s}_k$ is a sequence of k terms, for $0 \leq k \leq 1$).
3. Nothing else is an atomic formula.

These are atomic formulas with 1-place predicates ($k = 1$):

$$G^1(n) \quad P^1_0(a) \quad F^1_{613}(x)$$

Because atomic formulas with 0-place predicates ($k = 0$) have an empty sequence of terms, we omit their brackets and write:

$$A^0 \quad B^0_0 \quad H^0_{22}$$

This resembles SL's notation for letters. ML has none. But atomic formulas with 0-place predicates take their place.

Formulas are then inductively defined on the basis of atomic formulas according to the following rules:

An ML FORMULA is generated by these rules:

BASE	Every atomic formula is a formula.
INDUCTIVE	$\neg\mathbf{A}$ is a formula if $\mathbf{A}$ is.
	$(\mathbf{A} \wedge \mathbf{B})$ is a formula if $\mathbf{A}$,$\mathbf{B}$ are.
	$(\mathbf{A} \vee \mathbf{B})$ is a formula if $\mathbf{A}$,$\mathbf{B}$ are.
	$(\mathbf{A} \rightarrow \mathbf{B})$ is a formula if $\mathbf{A}$,$\mathbf{B}$ are.
	$(\mathbf{A} \leftrightarrow \mathbf{B})$ is a formula if $\mathbf{A}$,$\mathbf{B}$ are.
	$\exists\boldsymbol{\nu}\mathbf{A}$ is a formula if $\mathbf{A}$ is a quantifier-free formula and $\boldsymbol{\nu}$ is a variable.
	$\forall\boldsymbol{\nu}\mathbf{A}$ is a formula if $\mathbf{A}$ is a quantifier-free formula and $\boldsymbol{\nu}$ is a variable.
CLOSURE	Nothing else is a formula.

The BASE rule provides inputs for the INDUCTIVE rule, which specifies how connectives *and quantifiers* generate formulas from their inputs. The rules for connectives are analogus to those from SL. What's new concerns how *quantifier* formulas are generated from *quantifier-free* formulas and variables. Again, the CLOSURE rule says that these are the *only* ways to generate sentences.

As before, how a formula is generated is its *ancestry*. For example, the ancestry of $\neg(\forall x F(x) \vee R(c))$ is:

1. $F(x)$ and $R(c)$ are formulas	(BASE)
2. $\forall x F(x)$ is a formula	(INDUCTIVE on 1)
3. $\forall x F(x) \vee R(c)$ is a formula	(INDUCTIVE on 1,2)
4. $\neg(\forall x F(x) \vee R(c))$ is a formula	(INDUCTIVE on 3)

And, once again, a formula's ancestry is *unique*. Complex formulas have a main connective. Atomic formulas have none.

6.2.1 Substitution Instances

A **substitution instance** of a quantifier formula is a formula that results from removing its leading quantifier and replacing the variable it bound with a name:

A SUBSTITUTION INSTANCE of a quantifier formula $\exists \boldsymbol{v}\mathbf{A}(\ldots\boldsymbol{v}\ldots)$ or $\forall \boldsymbol{v}\mathbf{A}(\ldots\boldsymbol{v}\ldots)$ is the formula $\mathbf{A}(\ldots\mathbf{n}\ldots)$ that results by uniformly replacing every occurrence of $\boldsymbol{v}$ with name $\mathbf{n}$.

To illustrate, consider the sentence $\exists x(P(x) \vee \neg Q(x))$. This is a substitution instance of it:

$$P(c) \vee \neg Q(c)$$

But this is not a substitution instance of it:

$$P(c) \vee \neg Q(d)$$

Not all occurrences of x were uniformly replaced by the same name.

6.2.2 Scope

ML's notion of **scope** is like SL's (2.3) except it concerns what part of a *formula* a connective *or a quantifier* applies to:

The SCOPE of a *connective* in a formula is the formula that has it as its main connective. The SCOPE of a *quantifier* in a formula is the formula it prefixes.

To illustrate, consider:

$$\neg(\forall y G(y) \wedge F(x))$$

Its main connective is $\neg$. Its scope is the *entire* formula. The scope of the $\wedge$ is the formula:

$$\forall y G(y) \wedge F(x)$$

And the scope of the quantifier $\forall$ is the formula:

$$G(y)$$

By INDUCTIVE, a quantifier can only be prefixed to a *quantifier-free* formula. No quantifier can occur in the scope of another. Quantifiers don't nest. And so the expression:

$$\exists y \forall x(F(x) \rightarrow G(y))$$

is not a formula. This prohibition is a defining feature of ML. Lifting

it in FOL yields an enormous increase in power and complexity.

6.3 Sentences

All ML sentences are formulas, but not all formulas are sentences. It depends on whether the formula has **free variables**:

An ML SENTENCE is a formula with no free variables.

What is a **free variable**? It is a variable that is *not* **bound**:

A variable $\boldsymbol{v}$ is BOUND iff it is within the scope of a quantifier; otherwise it is FREE (or UNBOUND).

In ML, quantifiers are the *only* device for binding variables:

- Any variable in an atomic formula is free.
- Variables free in $\mathbf{A}$ are free in $\neg\mathbf{A}$.
- Variables free in $\mathbf{A}$,$\mathbf{B}$ are free in $(\mathbf{A} \wedge \mathbf{B})$, $(\mathbf{A} \vee \mathbf{B})$, $(\mathbf{A} \rightarrow \mathbf{B})$, and $(\mathbf{A} \leftrightarrow \mathbf{B})$.
- Variables free in $\mathbf{A}$ are free in $\forall\boldsymbol{v}\mathbf{A}$ and $\exists\boldsymbol{v}\mathbf{A}$, except for $\boldsymbol{v}$ which is bound by $\forall$ in $\forall\boldsymbol{v}\mathbf{A}$ and by $\exists$ in $\exists\boldsymbol{v}\mathbf{A}$.

In the following formula, x is not in the scope of any quantifiers because there are none:

$$\neg(G(a) \wedge F(x))$$

So, x is free. By contrast, the variable x is bound in this formula:

$$\neg(G(a) \wedge \exists x F(x))$$

The variable x is also bound in this formula:

$$\exists x \neg(G(a) \wedge F(x))$$

A formula may contain some bound variables and some free variables. In this example, y is bound while x is free:

$$\neg(\forall y G(y) \wedge F(x))$$

CHAPTER 7

Semantics

This chapter focuses on the formal semantics of ML.

7.1 Models

Because SL ignored subatomic structure, its valuations could assign semantic values directly to atomic sentences. But ML recognizes *subatomic* structure. So, it needs a richer semantic notion: the **model**.

A MODEL $\mathbb{M} = \langle \mathbb{D}, \mathbb{I} \rangle$, for domain $\mathbb{D}$ and interpretation $\mathbb{I}$.

We now specify what domains and interpretations are.

7.1.1 Domains

A **domain**, symbolized as '$\mathbb{D}$', is a set. Other logics allow it to be empty. Not ML. It can be finite or infinite, but must not be empty:

A DOMAIN $\mathbb{D}$ is any non-empty set.

Anything may be included. When we *apply* ML, we may limit what to include. But those limits give the application *content*. Our focus is on *form*. So, we won't limit what may go into a domain. When a domain is unspecified, it defaults to *everything*.

7.1.2 Interpretations

An **interpretation**, like a valuation, assigns semantic values to expressions. But it also takes into account some of ML's additional expressions: predicates and terms. (Quantifiers require special treatment.) We use the symbol '$\mathbb{I}$' for an interpretation.

An INTERPRETATION $\mathbb{I}$ on domain $\mathbb{D}$ assigns:

1. To each name $\boldsymbol{n}$, an item x from $\mathbb{D}$.
2. To each predicate $\boldsymbol{P}^k$ of degree k, a set of zero or more k-tuples of items from $\mathbb{D}$.

To specify an interpretation, specify its domain and whichever names and predicates are of interest (the rest may be omitted).

7.1.2.1 Names

An intepretation assigns to each name exactly one item from the domain. The name **refers** to this item, and it is its **referent**:

Name $\boldsymbol{n}$'s REFERENT is the domain item assigned to it.

An item can be nameless. But ML disallows *empty names* without referents. This interpretation displays the referents of names a, b:

$\mathbb{D}$: {5,6,7,8,9}
a: 5
b: 9

An item may have many names. If so, these names **corefer**:

Names $\boldsymbol{m}, \boldsymbol{n}$ COREFER iff $\boldsymbol{m}, \boldsymbol{n}$ have the same referent.

This interpretation makes a, b corefer to item 7:

$\mathbb{D}$: {5,6,7,8,9}
a: 7
b: 7

An interpretation might even make *all* names corefer.

7.1.2.2 Predicates

An interpretation assigns an **extension** to every predicate:

The EXTENSION of predicate $\boldsymbol{P}^k$ is a set of zero or more k-tuples of items from the domain.

To illustrate, consider a 1-place predicate F and this domain:

$\mathbb{D}$: {0,1,2,3,4,5,6,7,8,9}

One interpretation assigns this extension to F:

$$F\text{: } \{\langle 0\rangle, \langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 4\rangle\}$$

Other interpretations assign different extensions to F, even the empty set. But an interpretation can only assign sequences of items from the domain. Consider a domain without 0,1,2:

$$\mathbb{D}\text{: } \{3,4,5,6\}$$

The extension above *cannot* be assigned to F for *this* domain.

There are only two possible extensions to assign to a 0-place predicate: the set containing the empty sequence $\{\langle\rangle\}$ or the empty set $\emptyset$. These two sets differ because their members differ: the first has a member ($\langle\rangle$), the second is empty. These subtleties will allow us to treat 0-place predicates uniformly with the rest.

Predicates may have the same or different extensions. When their extensions are the same, they are **coextensive**:

> Predicates $\boldsymbol{P}$,$\boldsymbol{Q}$ are COEXTENSIVE iff $\boldsymbol{P}$,$\boldsymbol{Q}$ have the same extension.

7.1.3 Quantifiers

Were SL's valuations transposed to ML, we would assign 1 to an atomic sentence $\boldsymbol{P}(\boldsymbol{n})$ iff $\boldsymbol{n}$'s referent is in $\boldsymbol{P}$'s extension, and assign 0 otherwise. For semantic tables, we would replace letters with atomic sentences and calculate the semantic values of complex sentences as before. For example, the semantic table for $\neg F(a)$ would be:

$F(a)$	$\neg F(a)$	
1	0	1
0	1	0

Quantifiers dash our hope. Consider $\forall x F(x)$ and $\exists x F(x)$. These are a complex sentences that prefix a quantifier to the atomic formula $F(x)$. So, we'd expect a semantic table to look like:

$F(x)$	$\forall x\, F(x)$	
?	?	?

$F(x)$	$\exists x\, F(x)$	
?	?	?

There is a problem. Semantic tables display how a valuation assigns

a semantic value to each *sentence*. But $F(x)$ is *formula*, not a *sentence*. So, the table does not display a valuation. The semantics for quantifiers must be given in some other way.

The idea is to assign 1 to $\exists \boldsymbol{v}\mathbf{F}(\boldsymbol{v})$ when $\mathbf{F}(\boldsymbol{v})$ would get assigned 1 for *some* item in the domain, and assign 1 to $\forall \boldsymbol{v}\mathbf{F}(\boldsymbol{v})$ when $\mathbf{F}(\boldsymbol{v})$ would get assigned 1 for *all* items in the domain. The semantic value of a quantified formula binding a variable is thereby determined by the semantic value of a formula *without* that quantifier:

$\boldsymbol{v}$ is bound		$\boldsymbol{v}$ is free
$[\![\exists \boldsymbol{v}\mathbf{F}(\boldsymbol{v})]\!]$	determined by	$[\![\mathbf{F}(\boldsymbol{v})]\!]$
$[\![\forall \boldsymbol{v}\mathbf{F}(\boldsymbol{v})]\!]$	determined by	$[\![\mathbf{F}(\boldsymbol{v})]\!]$

We need to assign a semantic value of a formula with a *free* variable ($\boldsymbol{v}$). We use the notion of a **(variable) assignment**:

> An (VARIABLE) ASSIGNMENT α assigns an item from the domain to each variable.

For example, given a domain of all natural numbers $0, 1, 2, \ldots$, here are a few (partial) assignments over it:

	s	s_0	s_1	…	x	x_0	x_1	…	z	z_0	z_1	…
α_0 :	0	0	0	…	0	0	0	…	0	0	0	…
α_1 :	1	0	0	…	0	0	0	…	0	0	0	…
⋮				⋮				⋮				⋮
α_m :	1	0	0	…	0	0	6	…	0	8	0	…
α_n :	13	4	92	…	66	1	6	…	4	3	47	…
⋮				⋮				⋮				⋮

We may specify *which* item d is assigned to a given variable v by writing: $\alpha(v) = d$. So, for example, $\alpha_0(z) = 0$ and $\alpha_n(s_1) = 92$.

When two assignments assign the same item to the same variable, they *agree* on that variable:

> Assignments α_j, α_k AGREE on variable $\boldsymbol{v}$ iff $\alpha_j(\boldsymbol{v}) = \alpha_k(\boldsymbol{v})$; else, they DISAGREE on $\boldsymbol{v}$.

For example, α_0, α_m agree on x but disagree on x_1. We write $\alpha[\boldsymbol{d}/\boldsymbol{v}]$ to specify the assignment that disagrees with α *at most* by assigning $\boldsymbol{d}$ in $\mathbb{D}$ to $\boldsymbol{v}$. The "at most" matters. It *limits* disagreement without

requiring any. For example, $\alpha_0[1/s]$ at most disagrees with α_0 by assigning 1 to s. Which assignment is $\alpha_0[1/s]$? Neither α_m nor α_n: each disagrees with α_0 on more than just s. But α_1 assigns 1 to s and otherwise agrees with α_0. So, $\alpha_0[1/s] = \alpha_1$. Disagreement is *limited* but *not* required. For example, $\alpha_0[0/s]$ at most disagrees with α_0 by assigning 0 to s. But α_0 already assigns 0 to s. So, α_0 and $\alpha_0[0/s]$ do not disagree on s, or anywhere else. Trivially, they disagree *at most* on s because they do not disagree at all. So, $\alpha_0[0/s] = \alpha_0$.

7.2 Formal Semantics

ML's semantics is specified in stages for subatomic expressions, formulas, and sentences.

7.2.1 Subatomic Expressions

Semantic values for subatomic expressions are assigned relative to a model under an assignment as follows:

The SEMANTIC VALUE $[\![\cdot]\!]^{\alpha}_{\mathbb{M}}$ for model $\mathbb{M}$ under assignment α is:

S1 $[\![\boldsymbol{P}^k]\!]^{\alpha}_{\mathbb{M}} = \mathbb{I}(\boldsymbol{P}^k)$

S2 $[\![\boldsymbol{n}]\!]^{\alpha}_{\mathbb{M}} = \mathbb{I}(\boldsymbol{n})$

S3 $[\![\boldsymbol{v}]\!]^{\alpha}_{\mathbb{M}} = \alpha(\boldsymbol{v})$

Rules S1-S3 specify the semantic values for *subatomic* expressions. A predicate's semantic value is its extension (S1). A name's semantic value is its referent (S2). And a variable's semantic value is the item assigned to it by an assignment (S3).

7.2.2 Formulas

Semantic values for formulas are determined by the subatomic expressions composing them.

7.2.2.1 Atomic Formulas

Semantic values for atomic formulas are determined directly ($\top, \bot$) or by the subatomic expressions composing them:

The SEMANTIC VALUE $[\![\cdot]\!]^{\alpha}_{\mathbb{M}}$ for model $\mathbb{M}$ under assignment α is:

A1 $[\![\top]\!]^{\alpha}_{\mathbb{M}} = 1$

A2 $[\![\bot]\!]^{\alpha}_{\mathbb{M}} = 0$

A3 $[\![\boldsymbol{P}^k(\boldsymbol{t})]\!]^{\alpha}_{\mathbb{M}} = 1$ iff $\langle [\![\boldsymbol{t}]\!]^{\alpha}_{\mathbb{M}} \rangle \in [\![\boldsymbol{P}^k]\!]^{\alpha}_{\mathbb{M}}$

Rules A1-A3 specify semantic values for *atomic* formulas. The semantic values of $\top$ and $\bot$ are invariant (A1,A2). Rule A3 specifies the semantic value of an atomic formula: it is assigned 1 when the sequence of semantic values of its terms is a member of its predicate's extension. This sequence SATISFIES the formula. For example, the semantic value of $\boldsymbol{P}^1(\boldsymbol{t})$ is 1 when the 1-tuple of $\boldsymbol{t}$'s semantic value is in $\boldsymbol{P}^1$'s extension, else 0. This extends to 0-place predicates. Consider:

$\mathbb{D}$: {0,1,2,3,4,5,6,7,8,9}
A: $\{\langle\rangle\}$
B: { }

A 0-place predicate's extension must be the empty set {} or the set of the empty sequence $\{\langle\rangle\}$. The empty sequence $\langle\rangle$ *is* a member of $\{\langle\rangle\}$, but *not* of {}. So, $\{\langle\rangle\} \neq \{\}$ and A, B are not coextensive. Each must combine with a 0-tuple of names: $\langle\rangle$. It is in A's extension. So, A's semantic value is 1. Nothing is in B's extension, not even $\langle\rangle$. So, B's semantic value is 0. In effect, ML's semantics treats 0-degree predicates like letters, thus emulating SL's semantics.

7.2.2.2 Complex Formulas

Semantic values for complex formulas are determined by the subformulas composing them:

The SEMANTIC VALUE $[\![\cdot]\!]^{\alpha}_{\mathbb{M}}$ for model $\mathbb{M}$ under assignment α is:

C1 $[\![\neg \mathbf{F}]\!]^{\alpha}_{\mathbb{M}} = 1 - [\![\mathbf{F}]\!]^{\alpha}_{\mathbb{M}}$

C2 $[\![\mathbf{F} \vee \mathbf{G}]\!]^{\alpha}_{\mathbb{M}} = \max([\![\mathbf{F}]\!]^{\alpha}_{\mathbb{M}}, [\![\mathbf{G}]\!]^{\alpha}_{\mathbb{M}})$

C3 $[\![\mathbf{F} \wedge \mathbf{G}]\!]^{\alpha}_{\mathbb{M}} = \min([\![\mathbf{F}]\!]^{\alpha}_{\mathbb{M}}, [\![\mathbf{G}]\!]^{\alpha}_{\mathbb{M}})$

C4 $[\![\mathbf{F} \rightarrow \mathbf{G}]\!]^{\alpha}_{\mathbb{M}} = \max([\![\neg \mathbf{F}]\!]^{\alpha}_{\mathbb{M}}, [\![\mathbf{G}]\!]^{\alpha}_{\mathbb{M}})$

C5 $[\![\mathbf{F} \leftrightarrow \mathbf{G}]\!]^{\alpha}_{\mathbb{M}} = \min([\![\mathbf{F} \rightarrow \mathbf{G}]\!]^{\alpha}_{\mathbb{M}}, [\![\mathbf{G} \rightarrow \mathbf{F}]\!]^{\alpha}_{\mathbb{M}})$

C6 $[\![\exists \boldsymbol{v} \mathbf{F}]\!]^{\alpha}_{\mathbb{M}} = \max([\![\mathbf{F}]\!]^{\alpha[\boldsymbol{d}/\boldsymbol{v}]}_{\mathbb{M}})$ for all $\alpha[\boldsymbol{d}/\boldsymbol{v}]$

C7 $[\![\forall \boldsymbol{v} \mathbf{F}]\!]^{\alpha}_{\mathbb{M}} = \min([\![\mathbf{F}]\!]^{\alpha[\boldsymbol{d}/\boldsymbol{v}]}_{\mathbb{M}})$ for all $\alpha[\boldsymbol{d}/\boldsymbol{v}]$

Rules C1-C7 specify semantic values for *complex* formulas. Rules C1-C5 repeat those for SL's connectives. Rules C6-C7 are for quantifers:

$\exists \nu \mathbf{F}$ is assigned 1 if *some* item assigned to ν assigns 1 to $\mathbf{F}$, while $\forall \nu \mathbf{F}$ is assigned 1 if *every* item assigned to ν assigns 1 to $\mathbf{F}$.

7.2.3 Sentences

It is provable that a sentence assigned 1 in a model under *some* assignment will get assigned 1 for them *all*. So, we define the semantic value of a *sentence* relative just to a model:

> The SEMANTIC VALUE $[\![\cdot]\!]_{\mathbb{M}}$ for sentence $\mathbf{A}$ and model $\mathbb{M}$ is: $[\![\mathbf{A}]\!]_{\mathbb{M}} = 1$ iff $[\![\mathbf{A}]\!]^{\alpha}_{\mathbb{M}} = 1$ for some assignment α.

7.3 Semantic Concepts

ML's semantic concepts just replace valuations with models:

> Γ ENTAILS $\mathbf{C}$ iff there is no model $\mathbb{M}$ where $\min([\![\Gamma]\!]_{\mathbb{M}}) > [\![\mathbf{C}]\!]_{\mathbb{M}}$

As in SL, entailment in ML is **monotonic**:

> MONOTONICITY. If $\Gamma \vDash \mathbf{C}$, then $\mathbf{A}, \Gamma \vDash \mathbf{C}$ (for any $\mathbf{A}$).

Other concepts are defined as before in terms of entailment:

- $\mathbf{A}$ is a VALIDITY iff $\emptyset \vDash \mathbf{A}$ (or: $\vDash \mathbf{A}$).
- $\mathbf{A}$ is a CONTRADICTION iff $\mathbf{A} \vDash \bot$.
- $\mathbf{A}$ is a CONTINGENCY iff neither $\vDash \mathbf{A}$ nor $\mathbf{A} \vDash \bot$.
- $\mathbf{A}, \mathbf{B}$ are EQUIVALENT iff $\mathbf{A} \vDash \mathbf{B}$ and $\mathbf{B} \vDash \mathbf{A}$.
- Γ is SATISFIABLE iff $\Gamma \nvDash \bot$.

But our *tests* for determining whether the concepts apply must change. In SL, we checked semantic tables. These could only have a finite number of rows. In ML, we check models. They are infinite. Some tests only require checking one. But others require checking them all indirectly by considering what holds in an *arbitrary* model.

Entailment: show that an *arbitrary* model assigns 1 to the conclusion of an argument if it assigns 1 to all its premises. Does $\{F(a), G(a)\}$ entail $\exists x(F(x) \wedge G(x))$? Consider an arbitrary model that assigns 1 to $F(a)$ and $G(a)$. Then the semantic value of a is in both F's and G's extension. *Some* item satisfies both F and G, so $\exists x(F(x) \wedge G(x))$ is assigned 1. So, $\{F(a), G(a)\} \vDash \exists x(F(x) \wedge G(x))$.

***Non*entailment**: give a model that assigns all of an argument's premises 1 and its conclusion 0. This is a **countermodel** to the premises entailing the conclusion. Does $\{F(a), G(b)\}$ entail $\exists x(F(x) \wedge G(x))$? Consider:

$\mathbb{D}$: {6,23}
F: $\{\langle 6\rangle\}$
G: $\{\langle 23\rangle\}$
a: 6
b: 23

The semantic value of a is in F's extension and the semantic value of b is in G's extension. So, this model assigns 1 to both premises $F(a)$ and $G(b)$. But *no* item in the domain is in *both* of F's and G's extensions. So, the model assigns 0 to the conclusion $\exists x(F(x) \wedge G(x))$. And so $\{F(a), G(b)\} \nvDash \exists x(F(x) \wedge G(x))$.

7.3.1 Validity

Validity: show that a sentence is assigned 1 in an *arbitrary* model. Is $\forall x(A(x) \vee \neg A(x))$ a validity? In an arbitrary model, each item in its domain either will, or will not, be in A's extension. If an item is in A's extension, then it satisfies $A(x)$, and so $A(x) \vee \neg A(x)$. Or, if an item is not in A's extension, then it satisfies $\neg A(x)$, and so $A(x) \vee \neg A(x)$. Either way, each item satisfies $A(x) \vee \neg A(x)$. So, they all do. This reasoning only used generic facts about an arbitrary model $\mathbb{M}$. And so we may conclude that $\forall x(A(x) \vee \neg A(x))$ is a validity.

***Non*validity**: give a model that assigns a sentence 0. This is a **countermodel** to its validity. Is $F(a)$ a validity? Consider:

$\mathbb{D}$: {0,1}
F: $\{\langle 0\rangle\}$
a: 1

The semantic value of a is *not* in F's extension. So, we have a countermodel. $F(a)$ is *not* a validity.

7.3.2 Contradiction

Contradiction: show that a sentence is assigned 0 in an *arbitrary* model. Is $\exists x(A(x) \wedge \neg A(x))$ a contradiction? In an arbitrary model,

no item in its domain is *both* in and not in A's extension. So, *no* item satisfies $\exists x(A(x) \wedge \neg A(x))$. It is a contradiction.

***Non*contradiction**: give *a* model that assigns a sentence 1. (Confusingly, such a model "models" that sentence.) Is $F(a)$ a contradiction? Consider:

$\mathbb{D}$: {0,1}
F: {⟨0⟩}
a: 0

The semantic value of a *is* in F's extension. So, we have a model for $F(a)$. It is *not* a contradiction.

7.3.3 Contingency

Contingency: show that a sentence is neither a validity nor a contradiction. Use the tests above.

***Non*contingency**: show that a sentence is a validity or a contradiction. Again, use the tests above.

7.3.4 Equivalence

Equivalence: show that an *arbitrary* model assigns a pair of sentences the same semantic value. Are $\exists x A(x)$ and $\exists y A(y)$ equivalent? In an arbitrary model, A's extension is, or is not, empty. If empty, no item satisfies A. So, both $\exists x A(x)$ and $\exists y A(y)$ get assigned 0. If not empty, some item satisfies A. So, both $\exists x A(x)$ and $\exists y A(y)$ get assigned 1. Either way, both are assigned the same semantic values. So, they are equivalent.

More interestingly, $\forall x F(x)$ and $\neg\exists x\neg F(x)$ are equivalent. In an arbitrary model, $\forall x F(x)$ is assigned 1 only if every item is in F's extension. If all are, none is not. So, $\neg\exists x\neg F(x)$ is assigned 1 if $\forall x F(x)$ is. And, in an arbitrary model, $\neg\exists x\neg F(x)$ is assigned 1 only if no item is not in F's extension. If none are not, then all are. So, $\forall x F(x)$ is assigned 1 if $\neg\exists x\neg F(x)$ is. Similar reasoning shows that $\exists x F(x)$ and $\neg\forall x\neg F(x)$ are equivalent.

Inequivalence: give *a* model that assigns a pair of sentences different semantic values. This is a **countermodel** to their equivalence. Is the pair $F(a)$ and $\exists x\neg F(x)$ inequivalent? Consider:

$\mathbb{D}$: $\{0,1\}$
F: $\{\langle 0\rangle\}$
a: 1

The semantic value of a is *not* in F's extension, so $F(a)$ is assigned 0. *Some* item (1) is *not* in F's extension, so $\exists x \neg F(x)$ is assigned 1. We have a countermodel. The pair is inequivalent.

7.3.5 Satisfiability

Unsatisfiability: show that an *arbitrary* model cannot assign 1 to all sentences in a set. Is the set $\{\forall x P(x), \exists x \neg P(x)\}$ unsatisfiable? Suppose a model assigns 1 to $\forall x P(x)$. Then no item in its domain satisfies $\neg P(x)$. So, it assigns 0 to $\exists x \neg P(x)$. Now, suppose a model assigns 1 to $\exists x \neg P(x)$. Then some item is *not* in P's extension. So, it assigns 0 to $\forall x P(x)$. No model, then, can assign 1 to *all* members in the set. So, the set is unsatisfiable.

Satisfiability: give *a* model that assigns 1 to all sentences in a set. This is a **countermodel** to its unsatisfiability. Is the set $\{\forall x P(x), \exists x P(x)\}$ satisfiable? Consider:

$\mathbb{D}$: $\{0,1,2,3,4\}$
P: $\{\langle 0\rangle, \langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 4\rangle\}$

Some item in the domain satisfies P. So, $\exists x P(x)$ is assigned 1. Indeed, *all* items satisfy P. So, $\forall x P(x)$ is assigned 1. The model assigns 1 to *all* sentences in the set. So, the set is satisfiable.

CHAPTER 8

Derivations

This chapter focuses on derivations in ML.

8.1 Derivability

ML's derivational system extends SL's system: every SL rule is an ML rule too. Our focus will be on what's new: quantifier rules.

8.1.1 Introduction and Elimination Rules

We consider the introduction and elimination rules for the quantifiers $\forall, \exists$ in turn. Some are legitimate only if they satisfy certain conditions. Some new terms will help. Given a derivation from Γ to $\mathbf{C}$:

> Line m is UNDISCHARGED at line n iff (i) $0 \leq m \leq \#(\Gamma)$; or (ii) m is visible to n and a horizontal bar separates lines $m, m+1$.

The premises (if any) in Γ begin a derivation. '$\#(\Gamma)$' is the number of them. So, condition (i) ensures that every premise is undischarged at line n. Condition (ii) ensures that any visible subpremises are too.

> Name $\boldsymbol{n}$ is ARBITRARY at line m iff $\boldsymbol{n}$ is not in any lines undischarged at m.

8.1.1.1 Universal Quantifier

$\forall$I derives a universal generalization from an arbitrary instance:

i	$\mathbf{A}(\ldots\boldsymbol{n}\ldots)$		1. $\boldsymbol{n}$ is arbitrary at i
	$\forall\boldsymbol{v}\mathbf{A}(\ldots\boldsymbol{v}\ldots)$	$\forall$I i	2. $\boldsymbol{v}$ is not in $\mathbf{A}(\ldots\boldsymbol{n}\ldots)$

Conditions 1-2 ensure that a universal generalization is derivable only from an **arbitrary** instance.

∀E derives an instance of a universal generalization from it:

i	$\forall \boldsymbol{v}\mathbf{A}(\ldots\boldsymbol{v}\ldots)$	
	$\mathbf{A}(\ldots\boldsymbol{n}\ldots)$	∀E i

This derivation illustrates both rules:

1	$\forall x(F(x) \leftrightarrow G(x))$	
2	$F(b) \leftrightarrow G(b)$	∀E 1
3	$\quad G(b)$	
4	$\quad F(b)$	↔E 2, 3
5	$\quad F(b)$	
6	$\quad G(b)$	↔E 2, 5
7	$G(b) \leftrightarrow F(b)$	↔I 3–4, 5–6
8	$\forall y(G(y) \leftrightarrow F(y))$	∀I 7

The use of ∀I on line 9 to line 8 is legitimate only if it satisfies both conditions. Is b arbitrary at line 8? Yes. The only undischarged lines at line 8 are lines 1-2, and b is not in them. Is x in line 8? No. Both conditions are satisfied. The use of ∀I is legitimate.

8.1.1.2 Existential Quantifier

∃I derives an existential generalization from any instance:

i	$\mathbf{A}(\ldots\boldsymbol{n}\ldots)$	
	$\exists \boldsymbol{v}\mathbf{A}(\ldots\boldsymbol{v}\ldots)$	∃I i

∃E derives a sentence from an **arbitrary** instance of an existential generalization:

i	$\exists \boldsymbol{v}\mathbf{A}(\ldots\boldsymbol{v}\ldots)$		
j	$\quad \mathbf{A}(\ldots\boldsymbol{n}\ldots)$		1. $\boldsymbol{n}$ is arbitrary j
k	$\quad \mathbf{C}$		2. $\boldsymbol{v}$ is not in $\mathbf{A}(\ldots\boldsymbol{n}\ldots)$
	$\mathbf{C}$	∃E i, j–k	3. $\boldsymbol{v}$ is not in $\mathbf{C}$

Conditions 1-3 ensure that a sentence is derivable from an existential

generalization only if the subderivation uses an **arbitrary** instance.

This derivation illustrates both rules:

1	$\exists y(A(y) \wedge B(x))$	
2	$\quad A(e) \wedge B(e)$	
3	$\quad A(e)$	$\wedge$E 2
4	$\quad \exists z A(z)$	$\exists$I 3
5	$\exists z A(z)$	$\exists$E 1, 2–4

The use of $\exists$E on line 5 to lines 1,2-4 is legitimate only if it satisfies all three conditions. Is e arbitrary at 2? Yes. The only undischarged line at line 2 is line 1, and e is not in it. Is y in line 2? No. Is y in line 4? No. All three conditions are satisfied. The use of $\exists$E is legitimate.

8.1.2 Shortcut Rules

ML has **shortcut rules** for converting quantified forms:

m	$\forall x \neg \mathbf{A}$		m	$\neg \forall x \mathbf{A}$	
	$\neg \exists x \mathbf{A}$	CQ m		$\exists x \neg \mathbf{A}$	CQ m
m	$\exists x \neg \mathbf{A}$		m	$\neg \exists x \mathbf{A}$	
	$\neg \forall x \mathbf{A}$	CQ m		$\forall x \neg \mathbf{A}$	CQ m

8.2 Derivational Concepts

The derivational concepts familiar from SL carry over to ML:

A is a THEOREM iff $\vdash$ **A**
A, **B** are INTERDERIVABLE iff **A** $\vdash$ **B** and **B** $\vdash$ **A**
Γ is INCONSISTENT iff $\Gamma \vdash \bot$; else Γ is CONSISTENT
MONOTONICTY. If $\Gamma \vdash$ **C**, then **A**, $\Gamma \vdash$ **C** (for any **A**)

Finally, ML is both sound and complete, just as SL was:

SOUNDNESS. If $\Gamma \vdash$ **C**, then $\Gamma \vDash$ **C**
COMPLETENESS. If $\Gamma \vDash$ **C**, then $\Gamma \vdash$ **C**

CHAPTER 9

Applications

This chapter focuses on applying ML to natural languages.

9.1 Truth-conditional Semantics

Like with SL, our applications of ML interpret its semantics in terms of truth-values. But the interpretation is more complicated. Some of ML's formulas are *not* sentences. It is nonsense to interpret their semantic values as truth-values. An analogy shows why. Consider:

$$x^2 = 1$$

If we read this as an open formula, it is ambiguous between:

Existential	$x^2 = 1$, *for at least one* x
Universal	$x^2 = 1$, *for all* x

The existential reading is true (when x is -1 or 1). The universal reading is false (all other x). The ambiguous open formula itself cannot sensibly be true or false. Analogously, we cannot interpret semantic values for open formulas as truth-values. Even when a model under an assignment assigns 1 to an open formula, it is left ambiguous whether *some* or *all* assignments are at issue. Binding variables to quantifiers resolves the ambiguity by fixing whether it is *some* ($\exists$) or *all* ($\forall$). This "closes" the formula and makes it a sentence. So, we must only interpret semantic values of *sentences* as *truth-values*:

The SEMANTIC VALUE $[\![\cdot]\!]_{\mathbb{M}}$ for sentence $\mathbf{A}$ and model $\mathbb{M}$ is: $[\![\mathbf{A}]\!]_{\mathbb{M}} = \mathrm{T}$ iff $[\![\mathbf{A}]\!]^{\alpha}_{\mathbb{M}} = 1$ for some assignment α; else $[\![\mathbf{A}]\!]_{\mathbb{M}} = \mathrm{F}$.

9.1.1 Semantic Concepts

Reinterpreting the semantics is to reinterpret **entailment**:

Γ ENTAILS $\mathbf{C}$ iff there is no model $\mathbb{M}$ where $\min([\![\Gamma]\!]_{\mathbb{M}}) > [\![\mathbf{C}]\!]$

The new interpretation is: a set of premises entails a conclusion when there is no model in which the premises are true and the conclusion is false. The other concepts are defined as before. The semantic tests for them remain unchanged.

9.1.2 Derivations

Reinterpreting the semantics does not affect derivations.

9.2 Renderings

A **key** in ML specifies a domain, assigns items from the domain to names, and assigns relations to predicates. Just what a key represents, however, depends on what we want to render.

The familiar *grammatical* distinction between subject and predicate is a good (but imperfect) start. Consider:

Mike laughs.

'Mike' is the subject, 'laughs' is the predicate. This differs from:

Laughing is what Mike is doing.

The second focuses on what Mike is doing (laughing) more than the first. ML ignores such differences. So, they may be rendered the same. We may pick a key with a domain of people and Mike assigned to assigning Mike to name m:

$\mathbb{D}$: people
m: Mike

When assigning predicates, it helps to visualize "holes" where names go. Our sentence only has one name: 'Mike'. Deleting it yields:

$\_\_\_{}_1$ laughs

This yields a 1-place predicate L^1. When there are more slots, we must pair them with places in the relation. In this case, there's only one option: slot $_1$ is paired with the place for who's laughing now $\_\_\_{}_1$. We can then update our key to reflect this:

$\mathbb{D}$: people
m: Mike
$L^1(\_1)$: $\_\_\_1$ laughs

We used mnemonics: m for **M**ike and L^1 for **l**aughing. We don't have to: it could be p for Mike and Q^1 for laughing. But mnemonics are handy. Now, we render our sentence as:

$$L^1(m) \quad \Longleftarrow \quad \text{Mike laughs.}$$

We also allow deleting *zero* names. This results in *zero* "holes":

Mike laughs.

This just yields a 0-place predicate M^0. And that was our original sentence. (This illustrates why we identified letters with 0-place predicates.) We can update our key to reflect this:

$\mathbb{D}$: people
m: Mike
$L^1(\_1)$: $\_\_\_1$ laughs
$M^0(\cdot)$: Mike laughs

We now have two available renderings for our sentence:

$$\begin{aligned} L^1(m) \quad &\Longleftarrow \quad \text{Mike laughs.} \\ M^0 \quad &\Longleftarrow \quad \text{Mike laughs.} \end{aligned}$$

There is some sense in which these renderings "are the same". But there are *other* senses in which they are not. *Syntactically*, they differ: the first plugs m into slot $\_1$ whereas the second does no plugging in at all. And, *semantically*, they differ too: $L^1(m)$ is true only if $\langle m \rangle$ is in L^1's extension, whereas M^0 is true only if $\langle\rangle$ is in M^0's extension. These syntactic and semantic differences may not show up in ordinary English. But they show up in ML.

It is usually best to render with higher-degree predicates rather than lower-degree predicates. This allows us to capture more similarities in form. Consider these sentences together:

Mike laughs. Bob laughs.

They differ in *which* people they are about. But they are alike in ascribing the same property to each person: *is laughing*. Just how this

similarity is captured depends on our choice of predicate. Consider:

$\mathbb{D}$:	people
m:	Mike
b:	Bob
$L^1(\_1)$:	$\_\_\__1$ laughs
$M^0(\cdot)$:	Mike laughs
$B^0(\cdot)$:	Bob laughs

We could render the pair of sentences as:

M^0	$\Longleftarrow$	Mike laughs.
B^0	$\Longleftarrow$	Bob laughs.

But this obscures their similarity. A better rendering is:

$L^1(m)$	$\Longleftarrow$	Mike laughs.
$L^1(b)$	$\Longleftarrow$	Bob laughs.

This rendering displays that both English sentences share a form: both ascribe the same relationship (*is laughing*) to a person. Our main goal has been to display as much form as we reasonably can.

9.2.1 Domains

Renderings partly depend on the choice of domain. Consider:

The rag is blue, soft, and wet.

The domain could be everything. If so, we need separate predicates:

$\mathbb{D}$:	everything
r:	the rag
$B(\_1)$:	$\_\_\__1$ is blue
$S(\_1)$:	$\_\_\__1$ is soft
$W(\_1)$:	$\_\_\__1$ is wet

With this key, we may render the sentence as:

$$B(r) \wedge S(r) \wedge W(r) \quad \Longleftarrow \quad \text{The rag is blue, soft, and wet.}$$

But a key whose domain is restricted only to blue things would make the predicate B redundant:

$\mathbb{D}$: blue things
r: the rag
$S(\_1)$: $\_\_\_1$ is soft
$W(\_1)$: $\_\_\_1$ is wet

With this key, we may render the sentence as:

$$S(r) \wedge W(r) \quad \Longleftarrow \quad \text{The rag is blue, soft, and wet.}$$

And a key whose domain is restricted only to blue, soft things would make both predicates B and S redundant:

$\mathbb{D}$: blue, soft things
r: the rag
$W(\_1)$: $\_\_\_1$ is wet

With this key, we may render the sentence as:

$$W(r) \quad \Longleftarrow \quad \text{The rag is blue, soft, and wet.}$$

These examples illustrate how the choice of a domain can affect rendering. How to choose depends on how much form we want to represent. Our default is to represent as much as we can. So, the default domain is everything.

9.2.2 Quantities

Quantifiers are naturally rendered as expressing *quantities*: $\forall$ as 'for all' or 'every' and $\exists$ as 'some' or 'at least one'. Consider:

There are unicorns. Everything is a unicorn.

Let's use this key:

$\mathbb{D}$: everything
$U(\_1)$: $\_\_\_1$ is a unicorn

These sentences may be rendered as:

$$\exists x U(x) \quad \Longleftarrow \quad \text{There are unicorns.}$$
$$\forall x U(x) \quad \Longleftarrow \quad \text{Everything is a unicorn.}$$

The first says that *at least one* item in the domain is a unicorn,

whereas the second says that they *all* are unicorns.

One reason why the $\exists$ is called the '*existential* quantifier' is that it can express *existence*:

$$\exists x U(x) \quad \Longleftarrow \quad \text{Unicorns exist.}$$

Unicorns don't exist unless there is at least one unicorn. *Nonexistence* negates existence by saying there are *none*:

$$\neg\exists x U(x) \quad \Longleftarrow \quad \text{Unicorns do not exist.}$$

The order of the negation and quantifier matters. The difference between $\neg\exists$ and $\exists\neg$ is the difference between *nothing is so-and-so* and *something is not so-and-so*. For example, swapping the order says that some item is not a unicorn:

$$\exists x \neg U(x) \quad \Longleftarrow \quad \text{Some non-unicorns exists.}$$

Any *non-unicorn* would make this true. Not so for $\neg\exists x U(x)$: it requires there to be *zero unicorns*.

9.2.3 Restrictions

Quantifiers **quantify over** the domain. When the domain is everything, *anything* becomes relevant to the truth-value of a rendering. This can have unintended side-effects. One of these is to "scoop up" irrelevant items and make them relevant to a sentence's truth-value. Consider:

$\mathbb{D}$: everything
$B(\_1)$: $\_\_\_1$ is beer

Suppose we render 'There is beer' as:

$$\exists x B(x) \quad \Longleftarrow \quad \text{There is beer.}$$

This says that *some item in the domain* is beer. The domain is everything. So, everything is relevant, even distant locations. If there is beer *anywhere* in reality, then it makes 'There is beer' true. But that does not the answer the intended question when a guest asks, "Is there beer?". They want to know whether there is beer *in the cooler*. If the beer has run out, the answer "There is no beer" should

be correct. But not on our rendering:

$$\neg\exists x B(x) \quad \Longleftarrow \quad \text{There is no beer.}$$

This says that beer does not exist. That's false. And it misses that the intended question concerns what's in our cooler.

Rendering 'Everything is beer' as $\forall x B(x)$ says that *all items in the domain* are beer. The domain is everything, including the moon. So, the rendering says that the moon is beer. It isn't. So, 'Everything is beer' false. Still, it should be correct to answer "Everything is beer" if the cooler only contains beer.

All this illustrates how quantifying over everything can have unintended side-effects. But these can be avoided by *restricting our attention*. There are two ways to do this.

One way is to **restrict the domain** (9.2.1). We may restrict it to *what's in our cooler*. That excludes distant beer and the moon. Neither would be in the domain to unintentionally make the first sentence true and the second sentence false.

Another way is to **render the restriction**. The restriction is not imposed on the domain, but *within* the rendering by *symbolizing* it in ML. We may do so by adding a new predicate to our key:

$C(\__1)$: $\_\_\__1$ is in our cooler

The predicate C is only satisfied by what's in our cooler. Even if distant beers and the moon are in the domain, they are *not* in C's extension. This allows us to render 'There is beer in the cooler' as:

$$\exists x(B(x) \wedge C(x)) \quad \Longleftarrow \quad \text{There is beer in the cooler.}$$

This says that some item in the domain is *both* in the cooler *and* beer. In general, we use *conjuncts of a conjunction* to **restrict** $\exists$.

One might try to render 'Everything in the cooler is beer' as:

$$\forall x(C(x) \wedge B(x)) \quad \Longleftarrow \quad \text{Everything in the cooler is beer.}$$

But this says of each item in the domain that it is both in the cooler and beer. The moon is neither. So, the rendering is incorrect. A better one swaps the connective from $\wedge$ to $\rightarrow$:

$$\forall x(C(x) \rightarrow B(x)) \quad \Longleftarrow \quad \text{Everything is in the cooler and is beer.}$$

This says of each item in the domain that it is beer if it is in the cooler. The moon is not in the cooler. So, it is irrelevant whether or not it is beer. The rendering with $\rightarrow$ is correct. In general, we use the *antecedent of a conditional* to **restrict** $\forall$.

9.2.4 Indefinite Descriptions

An **indefinite description** is *ambiguous*: it says *something* is as described without specifying which. To illustrate, consider:

A politician spoke. A politician lied.

Each sentence decribes what *some* politician did without describing any *specific* politician. (Neither "name names".)

Indefinite descriptions may be rendered as existential quantifiers. To illustrate, use this key:

$\mathbb{D}$: everything
$P(\_1)$: $\_\_\_1$ is a politician
$S(\_1)$: $\_\_\_1$ spoke
$L(\_1)$: $\_\_\_1$ lied

Then we may use these renderings:

$$\exists x(P(x) \wedge S(x)) \quad \Longleftarrow \quad \text{A politician spoke.}$$
$$\exists x(P(x) \wedge L(x)) \quad \Longleftarrow \quad \text{A politician lied.}$$

Even if *some* item satisfies the first description and *some* item satisfies the second, it needn't be the *same* item. So, if we wanted to render the statement that a politican spoke *and* lied, we could by:

$$\exists x(P(x) \wedge S(x) \wedge L(x)) \quad \Longleftarrow \quad \text{A politician spoke and lied.}$$

The previous two renderings do not entail this, although they are entailed by it.

The negation of an indefinite description says that *nothing* satisfies the description. Consider:

$$\neg\exists x(P(x) \wedge L(x)) \quad \Longleftarrow \quad \text{No politician lied.}$$

This says that *nothing* satisfies the description of being a politician who lied.

9.3 Limitations

A key motivation for ML was to avoid SL's limitations (5.3). It does (9.3.1). But it also faces new limitations (9.3.2).

9.3.1 Limits Overcome

Recall that SL could not render this argument as valid:

Cy voted.
∴ Someone voted.

But ML can. Let us use this key:

$\mathbb{D}$: people
c: Cy
$V(\_1)$: $\_\_\__1$ voted

We may then render the argument as:

$V(c)$
$\therefore\ \exists x V(x)$

Premise $V(c)$ is true in a model only if the (1-tuple of) c's referent is in V's extension. If so, *some* item in the domain (c's referent) is in V's extension. And that makes the conclusion true. So, ML explains why the argument is valid.

It can also explain why this argument is valid.

Everyone voted.
∴ Someone voted.

Reusing the key above, we may render it as:

$\forall x V(x)$
$\therefore\ \exists x V(x)$

Premise $\forall x V(x)$ is true in a model only if every item in the domain satisfies V. But then *some* item satisfies V. So, the conclusion is true if the premise is true. So, ML explains why the argument is valid.

It can also explain why these sentences are equivalent:

Not everyone voted. Someone did not vote.

Reusing the key above, we may render them as:

$$\begin{array}{rcl} \neg\forall x V(x) & \Longleftarrow & \text{Not everyone voted.} \\ \exists x \neg V(x) & \Longleftarrow & \text{Someone did not vote.} \end{array}$$

If the first sentence is true, then not all items in the domain satisfy V. So, some item does not satisfy V. Then the second sentence is true. Conversely, if the second sentence is true, then some item does not satisfy V. So, not all items satisfy V. Then the first sentence is true. So, ML explains their equivalence.

It can also explain why these sentences are inconsistent:

Everyone voted. No one voted.

Reusing the key above yet again, we render these as:

$$\begin{array}{rcl} \forall x V(x) & \Longleftarrow & \text{Everyone voted.} \\ \neg\exists x V(x) & \Longleftarrow & \text{It is not the case that someone voted.} \end{array}$$

The set $\{\forall x V(x), \neg\exists x V(x)\}$ is inconsistent. Suppose both sentences are true in a model. The first is true only if every item in the domain satisfies V, whereas the second is true only if some item does *not*. So, there would have to be an item that does and does not satisfy V. That's a contradiction. So, ML explains why the set is inconsistent.

9.3.2 New Limitations

But ML has its own limitations. These stem from its disallowing many-place predicates and nested quantifiers.

To illustrate, focus on *identities*, as in:

$$\sqrt{4} = 2$$

ML has no good way to render identities. Identity is a *relationship* between items that holds when they are the same and not when they differ. Rendering a *relationship* requires predicates with *2 or more* places. ML disallows these. So, it cannot express relations, including identity. This means it has no way to render several familiar forms of argument. One of these is "swapping equals for equals", as in:

2 is even.
$\sqrt{4} = 2$
$\therefore$ $\sqrt{4}$ is even.

Another is "inferring nonidentity from difference", as in:

2 is even.
3 is not even.
$\therefore$ $3 \neq 2$

ML cannot render relations. So, it cannot render these arguments.

This limitation is not confined to identity. Consider this argument:

Mike punched Bob.
$\therefore$ Someone punched someone.

The most natural way to render this would render 'punched' as a 2-place predicate, using a key something like this:

$\mathbb{D}$: everything
m: Mike
b: Bob
$P(\_1, \_2)$: $\_\_\__1$ punched $\_\_\__2$

The argument could then be rendered as:

$P(m,b)$ $\Longleftarrow$ Mike punched Bob.
$\exists x \exists y P(x,y)$ $\Longleftarrow$ Someone punched someone.

But none of this makes sense in ML. Again, ML disallows 2-place predicates and nested quantifiers. So, neither the premise nor the conclusion are even formulas of ML.

The preceding limitations stem from ML's disallowing multi-placed predicates and nested quantifiers. These limitations do not detract from how nicely ML captures *truth-functional* and *monadic* forms. But they show that there *other forms* that ML does *not* capture. This motivates introducing a richer formal language that can capture these forms.

PART IV

First-order Form

CHAPTER 10
Syntax

This chapter focuses on the syntax of a formal language, FOL, capturing first-order form.

10.1 Expressions

FOL is a strict expansion of ML. Every ML expression is an FOL expression. But not all FOL expressions are ML expressions. FOL adds **many-place predicates** and **nested quantifiers**:

The ALPHABET OF FOL consists in:

TERMS	**Names**	$a, b, \ldots, r, \ldots, a_8, \ldots, r_8, \ldots$
	Variables	$s, t, u, v, w, x, y, z, s_0, \ldots, z_0, \ldots$
PREDICATES	**0-place**	$A^0, B^0, \ldots, Z^0, A^0_0, B^0_0, \ldots, Z^0_0, \ldots$
		$\top, \bot$
	1-place	$A^1, B^1, \ldots, Z^1, A^1_0, B^1_0, \ldots, Z^1_0, \ldots$
	2-place	$A^2, B^2, \ldots, Z^2, A^2_0, B^2_0, \ldots, Z^2_0, \ldots$
		$=$
	$\vdots$	$\vdots$
QUANTIFIERS		$\exists, \forall$
CONNECTIVES		$\neg, \vee, \wedge, \rightarrow, \leftrightarrow$
PUNCTUATION		$(\ ,\ )$

We define an expression in terms of the simple expressions:

An FOL EXPRESSION is any sequence from its alphabet.

As with ML, 0-place predicates stand in for letters. We may omit their superscripts: 'A' abbreviates 'A^0'.

Adding many-place predicates and nested quantifiers create new syntactic opportunities that give FOL immense power.

10.1.1 Predicates

FOL lifts ML's restriction to predicates with just 0 or 1 "slots". A predicate may be of *any* finite degree. For example, a 17-place predicate has degree 17. As before, uppercase letters $A, \ldots, Z$ (subscripted as needed) are for predicates. We use a superscript for a predicate's degree. So, F^1 is a 1-place predicate and Q_7^{33} is a 33-place predicate.

10.1.1.1 Identity

The 2-place predicate for **identity** gets special treatment. Its syntax is not special: it is syntactically like other 2-place predicates. Its semantics will make it special: it will get a fixed interpretation. We reserve the familiar symbol '=' for identity.

10.2 Formulas

Like ML, FOL's atomic sentences have internal structure characterized by the intermediate notion of a **formula**. We first define FOL formulas and then FOL sentences in terms of them.

An FOL ATOMIC FORMULA is generated by these rules:

1. $\top$ and $\bot$ are atomic formulas.
2. Every $\ulcorner \boldsymbol{P}^k(\mathbf{s_k})\urcorner$ is an atomic formula (where $\mathbf{s}_k$ is a string of k terms).
3. Nothing else is an atomic formula.

FOL expands on ML by allowing higher-degree predicates. A k-place predicate must combine with a k-tuple of terms. These are atomic formulas for 2-place predicates ($k = 2$):

$$G^2(n,t) \quad P_0^2(a,v) \quad F_{613}^2(x,y)$$

These are atomic formulas for predicates of degree 3 ($k = 3$):

$$A^3(a,b,c) \quad B_0^3(t,x,x) \quad H_{22}^3(z,y,x)$$

Our notation $P^k(n_1,\ldots,n_k)$ redundantly shows a predicate's degree by the superscript and the number of terms. A lone predicate without a superscript is ambiguous. To illustrate, P is ambiguous between 1-place P^1 and 2-place P^2. But a predicate can only occur in a formula

when combined with the right number of terms. For example, P^1 must combine with a 1-tuple and P^2 with a 2-tuple. So, P must be read as P^1 in $P(a)$ while P must be read as P^2 as in $P(a,b)$. Superscripts may be omitted when context disambiguates in this way.

Because '=' is a 2-place predicate, we could write $=(a,b)$. But we use the more familiar $a = b$. Strictly speaking, its negation should be $\neg(a = b)$. Again, we use the more familar: $a \neq b$.

Formulas are inductively defined on the basis of atomic formulas according to the following rules:

An FOL FORMULA is generated by these rules:

BASE	Every atomic formula is a formula.
INDUCTIVE	$\neg\mathbf{A}$ is a formula if $\mathbf{A}$ is.
	$(\mathbf{A} \wedge \mathbf{B})$ is a formula if $\mathbf{A}$,$\mathbf{B}$ are.
	$(\mathbf{A} \vee \mathbf{B})$ is a formula if $\mathbf{A}$,$\mathbf{B}$ are.
	$(\mathbf{A} \rightarrow \mathbf{B})$ is a formula if $\mathbf{A}$,$\mathbf{B}$ are.
	$(\mathbf{A} \leftrightarrow \mathbf{B})$ is a formula if $\mathbf{A}$,$\mathbf{B}$ are.
	$\exists\boldsymbol{\nu}\mathbf{A}$ is a formula if $\mathbf{A}$ is a $\boldsymbol{\nu}$-quantifier-free formula and $\boldsymbol{\nu}$ is a variable.
	$\forall\boldsymbol{\nu}\mathbf{A}$ is a formula if $\mathbf{A}$ is a $\boldsymbol{\nu}$-quantifier-free formula and $\boldsymbol{\nu}$ is a variable.
CLOSURE	Nothing else is a formula.

The BASE rule provides inputs for the INDUCTIVE rule, which specifies how connectives *and quantifiers* generate formulas from their inputs. The rules for connectives are analogus to those from SL. And the rules for *quantifier* formulas are *nearly* the same as those from ML. The only difference is that a $\boldsymbol{\nu}$-quantifier formula may now be generated from *any* formula that does not already contain $\boldsymbol{\nu}$. Again, the CLOSURE rule is crucial: the *only* way to generate sentences is by the BASE and INDUCTIVE rules.

We continue thinking of how a formula is generated as its *ancestry*. For example, the ancestry of $\neg(\forall x F(x) \vee R(c))$ is:

1. $F(x)$ and $R(y)$ are formulas	(BASE)
2. $F(x) \vee R(y)$ is a formula	(INDUCTIVE on 1)
3. $\forall x(F(x) \vee R(y))$ is a formula	(INDUCTIVE on 2)
4. $\neg\forall x(F(x) \vee R(y))$ is a formula	(INDUCTIVE on 3)
5. $\exists y\neg\forall x(F(x) \vee R(y))$ is a formula	(INDUCTIVE on 4)

Once again, a formula's ancestry is *unique*. Complex formulas FOL have a main connective. Atomic formulas have none.

10.3 Scope

FOL's notion of **scope** builds on ML's (6.2.2) and SL's (2.3):

> The SCOPE of a *connective* in a formula is the formula that has it as its main connective. The SCOPE of a *quantifier* in a formula is the formula it prefixes.

But FOL's syntactic additions allow for a new complication disallowed by ML: **nested quantifiers**. This is a quantifier within the scope of another quantifier. To illustrate, consider:

$$\exists x(\forall y R(y,x) \wedge F(x))$$

Its main connective is $\exists$. Its scope is the *entire* formula. The scope of the $\wedge$ is the formula:

$$\forall y R(y,x) \wedge F(x)$$

And the scope of the quantifier $\forall$ is the formula:

$$R(y,x)$$

The original formula was a sentence. Each subsequent formula, however, had a free variable. None of them are sentences.

10.4 Sentences

FOL sentences are defined as in ML(6.3). All sentences are formulas. But not all formulas are sentences. No *free* variables may occur in a sentence:

> An FOL SENTENCE is a formula with no free variables.

Once again, because all sentences are formulas, the definition of scope applies straightforwardly to sentences as well.

CHAPTER 11

Semantics

This chapter focuses on the formal semantics of FOL.

FOL's syntactic complexity vastly increases FOL's power but at the cost of losing **decidability**. But FOL's semantics is much like ML's. So, we focus on what's new.

11.1 Models

As in ML, an FOL **model** pairs a domain and interpretation:

> A MODEL $\mathbb{M} = \langle \mathfrak{D}, \mathbb{I} \rangle$, for domain $\mathbb{D}$ and interpretation $\mathbb{I}$.

11.1.1 Domains

A **domain** is just as it was in ML (7.1.1):

> A DOMAIN $\mathbb{D}$ is any non-empty set.

11.1.2 Interpretations

An **interpretation** is much like it was in ML (7.1.2):

> An INTERPRETATION $\mathbb{I}$ on domain $\mathbb{D}$ assigns:
> 1. To each name $\boldsymbol{n}$, some item x from $\mathbb{D}$.
> 2. To each k-place predicate $\boldsymbol{P}^k$, some set of zero or more k-tuples of items from $\mathbb{D}$.

Names are interpreted just as they were in ML(7.1.2.1).

Predicates are interpreted mostly as in ML (7.1.2.2). But FOL allows predicates of *any* nonnegative degree. This means adjusting which extensions can be assigned to them. To illustrate, here is a do-

main and interpretation that assigns extensions to the 3-place predicate F and the 10-place predicate G:

$\mathbb{D}$: {0,1,2,3,4,5,6,7,8,9}
F: {⟨0,1,2⟩, ⟨1,2,3⟩, ⟨2,3,4⟩, ⟨3,4,5⟩, ⟨4,5,0⟩}
G: {⟨0,1,2,3,4,5,6,7,8,9⟩}

11.1.3 Quantifiers

Quantifiers are interpreted mostly like they were in ML (7.1.3). But allowing nested quantifiers means keeping track of multiple variables in order. We'll return to this.

11.2 Formal Semantics

FOL's semantics mostly follows ML's 7. As before, we specify it in stage: subatomic expressions, formulas, and sentences.

11.2.1 Subatomic Expressions

Subatomic expressions get semantic values as in ML (7.2.1):

The SEMANTIC VALUE $[\![\cdot]\!]^{\alpha}_{\mathbb{M}}$ for model $\mathbb{M}$ under assignment α is:
S1 $[\![\boldsymbol{P}^k]\!]^{\alpha}_{\mathbb{M}} = \mathbb{I}(\boldsymbol{P}^k)$
S2 $[\![\boldsymbol{n}]\!]^{\alpha}_{\mathbb{M}} = \mathbb{I}(\boldsymbol{n})$
S3 $[\![\boldsymbol{v}]\!]^{\alpha}_{\mathbb{M}} = \alpha(\boldsymbol{v})$

11.2.2 Formulas

Much remains as it was in ML (7.2.2). Semantic values are still assigned to formulas relative to a model under an assignment, and a formula is still **satisfied** when it is assigned 1. The differences concern identities and nested quantifiers.

11.2.2.1 Atomic Formulas

Atomic formulas are assigned as in ML plus a clause for =:

The SEMANTIC VALUE $[\![\cdot]\!]^{\alpha}_{\mathbb{M}}$ for model $\mathbb{M}$ under assignment α is:

A1 $[\![\top]\!]^{\alpha}_{\mathbb{M}} = 1$
A2 $[\![\bot]\!]^{\alpha}_{\mathbb{M}} = 0$
A3 $[\![\boldsymbol{P}^k(\boldsymbol{t})]\!]^{\alpha}_{\mathbb{M}} = 1$ iff $\langle [\![\boldsymbol{t}]\!]^{\alpha}_{\mathbb{M}} \rangle \in [\![\boldsymbol{P}^k]\!]^{\alpha}_{\mathbb{M}}$
A4 $[\![\boldsymbol{m} = \boldsymbol{n}]\!]^{\alpha}_{\mathbb{M}} = 1$ iff $[\![\boldsymbol{m}]\!]^{\alpha}_{\mathbb{M}} = [\![\boldsymbol{n}]\!]^{\alpha}_{\mathbb{M}}$

11.2.2.2 Complex Formulas

Complex formulas are assigned just as in ML:

The SEMANTIC VALUE $[\![\cdot]\!]^{\alpha}_{\mathbb{M}}$ for model $\mathbb{M}$ under assignment α is:

C1 $[\![\neg \mathbf{F}]\!]^{\alpha}_{\mathbb{M}} = 1 - [\![\mathbf{F}]\!]^{\alpha}_{\mathbb{M}}$
C2 $[\![\mathbf{F} \vee \mathbf{G}]\!]^{\alpha}_{\mathbb{M}} = \max([\![\mathbf{F}]\!]^{\alpha}_{\mathbb{M}}, [\![\mathbf{G}]\!]^{\alpha}_{\mathbb{M}})$
C3 $[\![\mathbf{F} \wedge \mathbf{G}]\!]^{\alpha}_{\mathbb{M}} = \min([\![\mathbf{F}]\!]^{\alpha}_{\mathbb{M}}, [\![\mathbf{G}]\!]^{\alpha}_{\mathbb{M}})$
C4 $[\![\mathbf{F} \rightarrow \mathbf{G}]\!]^{\alpha}_{\mathbb{M}} = \max([\![\neg \mathbf{F}]\!]^{\alpha}_{\mathbb{M}}, [\![\mathbf{G}]\!]^{\alpha}_{\mathbb{M}})$
C5 $[\![\mathbf{F} \leftrightarrow \mathbf{G}]\!]^{\alpha}_{\mathbb{M}} = \min([\![\mathbf{F} \rightarrow \mathbf{G}]\!]^{\alpha}_{\mathbb{M}}, [\![\mathbf{G} \rightarrow \mathbf{F}]\!]^{\alpha}_{\mathbb{M}})$
C6 $[\![\exists \boldsymbol{v} \mathbf{F}]\!]^{\alpha}_{\mathbb{M}} = \max([\![\mathbf{F}]\!]^{\alpha[\boldsymbol{d}/\boldsymbol{v}]}_{\mathbb{M}})$ for all $\alpha[\boldsymbol{d}/\boldsymbol{v}]$
C7 $[\![\forall \boldsymbol{v} \mathbf{F}]\!]^{\alpha}_{\mathbb{M}} = \min([\![\mathbf{F}]\!]^{\alpha[\boldsymbol{d}/\boldsymbol{v}]}_{\mathbb{M}})$ for all $\alpha[\boldsymbol{d}/\boldsymbol{v}]$

But nested quantifiers complicate things. Consider:

$$\forall x \exists y L(x,y) \qquad \exists x \forall y L(x,y)$$

Order matters! The left formula $\forall x \exists y L(x,y)$ is assigned 1 just when putting *every* item for x assigns 1 to:

$$\exists y L(x,y)$$

And that happens just when, given the item put for x, putting *some* item for y assigns 1 to:

$$L(x,y)$$

By contrast, the right formula $\exists x \forall y L(x,y)$ is assigned 1 just when putting *some* item for x assigns 1 to:

$$\forall y L(x,y)$$

And that happens just when, given the item put for x, putting *every* item for y assigns 1 to:

$$L(x,y)$$

To illustrate how order matters, consider this model:

$\mathbb{D}$: $\{0,1,2\}$
L: $\{\langle 0,1\rangle, \langle 1,2\rangle, \langle 2,2\rangle\}$

To answer whether this model assigns 1 to $\forall x \exists y L(x,y)$, we cycle through *every* item putting it for x and *some* item for y and check whether that satisfies $L(x,y)$. We chart this with $x-$ and $y-$columns listing items put for x and y, an $\langle x,y\rangle$-column for their 2-tuples, and a $L(x,y)$-column for the semantic values assigned to $L(x,y)$ for that 2-tuple. Consider the cycle putting 0 for x:

x	y	$\langle x,y\rangle$	$L(x,y)$
0	0	$\langle 0,0\rangle$	0
0	1	$\langle 0,1\rangle$	1
0	2	$\langle 0,2\rangle$	0

When 0 is put for x there is *some* item (1) when, put for y, yields a 2-tuple ($\langle 0,1\rangle$) in L's extension. We're not done. We must cycle through *every* item put for x. So, extend the chart by putting 1 for x:

x	y	$\langle x,y\rangle$	$L(x,y)$
1	0	$\langle 1,0\rangle$	0
1	1	$\langle 1,1\rangle$	0
1	2	$\langle 1,2\rangle$	1

When 1 is put for x, there is *some* item (2) that when, put for y, yields a 2-tuple ($\langle 1,2\rangle$) in L's extension. To check the final case, extend the chart by putting 2 for x:

x	y	$\langle x,y\rangle$	$L(x,y)$
2	0	$\langle 2,0\rangle$	0
2	1	$\langle 2,1\rangle$	0
2	2	$\langle 2,2\rangle$	1

When 2 is put for x, there is *some* item (2) that when, put for y, yields a 2-tuple ($\langle 2,2\rangle$) in L's extension. So, for *every* item put for x there is *some* item that, when put for y, yields a 2-tuple in L's extension. And so the model satisfies the formula $\forall x \exists y L(x,y)$.

But does this model also satsify $\exists x \forall y L(x,y)$? To answer, check whether putting *some* item for x and *every* item for y would satisfy $L(x,y)$. Combining the previous three charts into one:

x	y	$\langle x,y\rangle$	$L(x,y)$
0	0	$\langle 0,0\rangle$	0
0	1	$\langle 0,1\rangle$	1
0	2	$\langle 0,2\rangle$	0
1	0	$\langle 1,0\rangle$	0
1	1	$\langle 1,1\rangle$	0
1	2	$\langle 1,2\rangle$	1
2	0	$\langle 2,0\rangle$	0
2	1	$\langle 2,1\rangle$	0
2	2	$\langle 2,2\rangle$	1

Now, we're asking whether there is *some* item when put for x and when *every* item is put for y yields a 2-tuple in L's extension? Rows 1,3 rule out item 0. Rows 4-5 rule out item 1. And rows 7-8 rule out item 2. That rules out every item in the domain. And so the model does *not* satisfy the formula $\exists x \forall y L(x,y)$.

It also matters which variables are bound to which quantifiers. To illustrate, compare $\exists y \forall x L(x,y)$ and $\exists x \forall y L(x,y)$. The only difference is that which variable each quantifier binds is swapped. Here is a model that assigns these sentences different sematic values:

$\mathbb{D}$: {0, 1}
L: {⟨0,0⟩, ⟨0,1⟩}

Some item (0), when assigned to x, satisfies $\forall y L(x,y)$. So, $\exists x \forall y L(x,y)$ gets assigned 1. By contrast, not every item (not 1), when assigned to y, satisfies $\exists x L(x,y)$. So, the $\exists y \forall x L(x,y)$ gets assigned 0.

11.2.2.3 Sentences

Semantic values of sentences are assigned just as in ML (7.2.3):

> The SEMANTIC VALUE $[\![\cdot]\!]_{\mathbb{M}}$ for sentence **A** and model $\mathbb{M}$ is: $[\![\mathbf{A}]\!]_{\mathbb{M}} = 1$ iff $[\![\mathbf{A}]\!]^{\alpha}_{\mathbb{M}} = 1$ for some assignment α.

11.3 Semantic Concepts

ML's semantic concepts and methods (7.3) transpose exactly to FOL:

> Γ ENTAILS **C** ($\Gamma \vDash \mathbf{C}$) iff $\min([\![\Gamma]\!]_{\mathbb{M}}) \leq [\![\mathbf{C}]\!]_{\mathbb{M}}$ for all $\mathbb{M}$

As in ML, entailment in FOL is **monotonic**:

MONOTONICITY. If $\Gamma \models \mathbf{C}$, then $\mathbf{A}, \Gamma \models \mathbf{C}$ (for any $\mathbf{A}$).

Other concepts are defined as before in terms of entailment:

- $\mathbf{A}$ is a VALIDITY iff $\emptyset \models \mathbf{A}$ (or: $\models \mathbf{A}$).
- $\mathbf{A}$ is a CONTRADICTION iff $\mathbf{A} \models \bot$.
- $\mathbf{A}$ is a CONTINGENCY iff neither $\models \mathbf{A}$ nor $\mathbf{A} \models \bot$.
- $\mathbf{A}, \mathbf{B}$ are EQUIVALENT iff $\mathbf{A} \models \mathbf{B}$ and $\mathbf{B} \models \mathbf{A}$.
- Γ is SATISFIABLE iff $\Gamma \not\models \bot$.

The tests for these concepts remain unchanged.

Entailment: show that an *arbitrary* model assigns 1 to the conclusion of an argument if it assigns 1 to all its premises. Does $\{\exists y \forall x L(x,y)\}$ entail $\forall x \exists y L(x,y)$? Consider an arbitrary model that assigns 1 to the premise. This requires that some item, when assigned to y, satisfies $\forall x L(x,y)$. If so, then every item, when assigned to x, satisfies $\exists y L(x,y)$. And so $\{\exists y \forall x L(x,y)\} \models \forall x \exists y L(x,y)$.

***Non*entailment**: give *a* model that assigns all of an argument's premises 1 and its conclusion 0. Does $\{\exists x \forall y L(x,y)\}$ entail $\forall x \exists y L(x,y)$? Consider:

$\mathbb{D}$: $\{0, 1\}$
L: $\{\langle 0,0\rangle, \langle 0,1\rangle\}$

Some item (0), when assigned to x, satisfies $\forall y L(x,y)$. So, the premise is assigned 1. But not every item (not 1), when assigned to x, satisfies $\exists y L(x,y)$. So, the conclusion is assigned 0. We have a countermodel. So, $\{\exists x \forall y L(x,y)\} \not\models \forall x \exists y L(x,y)$.

The two examples together show that $\forall x \exists y L(x,y)$ is a consequence of $\{\exists y \forall x L(x,y)\}$ but *not* of $\{\exists x \forall y L(x,y)\}$. It matters which quantifier binds which variable.

11.3.1 Validity

Validity: show that a sentence is assigned 1 in an *arbitrary* model. Is $\forall x \exists y (x = y)$ a validity? In an arbitrary model, each item will be identical to *some* item: itself. And so $\forall x \exists y (x = y)$ is a validity.

***Non*validity**: give *a* model that assigns a sentence 0. Is $\exists x \forall y (x = y)$ a validity? This model assigns it 1:

$\mathbb{D}$: $\{0\}$

This is because some item is identical to all items. But only because there is just one item that, trivially, is identical to them "all" by being identical to itself. Consider another model:

$\mathbb{D}$: $\{0, 1\}$

No item is identical to all items (0 is 0 but not 1; 1 is 1 but not 0). So, we have a countermodel. $\exists x \forall y (x = y)$ is *not* a validity.

11.3.2 Contradiction

Contradiction: show that a sentence is assigned 0 in an *arbitrary* model. Is $\forall x F(x) \wedge \exists x \neg F(x)$ a contradiction? Consider an arbitrary model that assigns 1 to the left conjunct $\forall x F(x)$. Then every item in the domain is in F's extension. But then *no* item is *not* in F's extension. So, the right conjunct $\exists x \neg F(x)$ is assigned 0. And so every model assigns the sentence 0. It is a contradiction.

***Non*contradiction**: give *a* model that assigns a sentence 1. Is $\forall x (R(x) \rightarrow x \neq x)$ a contradiction? This model assigns it 0:

$\mathbb{D}$: $\{0,1,2\}$
R: $\{\langle 0 \rangle\}$

This is because not all items satisfy $R(x) \rightarrow x \neq x$: 0 is in R's extension but is identical to itself. Consider another model:

$\mathbb{D}$: $\{0,1,2\}$
R: $\{\ \}$

In this model, every item satisfies $R(x) \rightarrow x \neq x$: no item in R's extension is not self-identical, but only because no item is in R's extension. So, this model assigns the sentence 1. It is *not* a contradiction.

11.3.3 Contingency

Contingency: show that a sentence is neither a validity nor a contradiction. Use the tests above.

***Non*contingency**: show that a sentence is a validity or a contradiction. Again, use the tests above.

11.3.4 Equivalence

Equivalence: show that an *arbitrary* model assigns a pair of sentences the same semantic value. Are $\exists x \exists y(x = y)$ and $\neg\forall x \forall y(x \neq y)$ equivalent? In an arbitrary model, its domain contains a self-identical item. So, all models assign 1 to $\exists x \exists y(x = y)$. Will any assign 0 to $\neg\forall x \forall y(x \neq y)$? Only if it assigns 1 to $\forall x \forall y(x \neq y)$. It can't do that if even one item, when assigned to x, *fails* to satisfy $\forall y(x \neq y)$. But every item is self-identical. All fail to satisfy $\forall y(x \neq y)$. So, all mdoels assign 1 to $\neg\forall x \forall y(x \neq y)$. The pair is equivalent.

Inequivalence: give *a* model that assigns a pair of sentences different semantic values. The countermodel given earlier to establish $\{\exists x \forall y L(x,y)\} \nvDash \forall x \exists y L(x,y)$ also shows that $\exists x \forall y L(x,y)$ and $\forall x \exists y L(x,y)$ are inequivalent.

11.3.5 Satisfiable

Unsatisfiability: show that an *arbitrary* model cannot assign 1 to all sentences in a set. Is the set $\{a = b, \neg R(a,b), R(b,b)\}$ unsatisfiable? Suppose a model assigns 1 to $a = b$. Then a, b co-refer to the same item. $R(a,b)$ is assigned 1 only if *that item*, when assigned to a and b, satisfies $R(a,b)$. And $R(b,b)$ is assigned 1 only if *that item*, when assigned to b, satisfies $R(b,b)$. So, $R(a,b)$ and $R(b,b)$ must get assigned the same semantic value. But then no model can assign 1 to both $\neg R(a,b)$ and $R(b,b)$. So, the set is unsatisfiable.

Satisfiability: give *a* model that assigns 1 to all sentences in a set. Is the set $\{\forall x(R(x) \rightarrow \neg\exists y(x = y)), \neg\exists x R(x)\}$ satisfiable? Consider:

$\mathbb{D}$: $\{0\}$
R: $\{\ \}$

No item is in R's extension. So, $\neg\exists x R(x)$ is assigned 1. Because no item is in R's extension, no item when assigned to x to satisfies $R(x) \rightarrow \neg\exists y(x = y)$. So, $\forall x(R(x) \rightarrow \neg\exists y(x = y))$ is also assigned 1. The set is satisfiable.

CHAPTER 12

Derivations

This chapter focuses on derivations in FOL.

12.1 Derivability

The derivational system for FOL strictly extends ML's system (and SL's). Every rule in ML's system (and so SL's) is a rule in FOL's system. We focus on what's new: rules for identity.

12.1.1 Introduction and Elimination Rules

We consider the introduction and elimination rules for '=' in turn.

12.1.1.1 Identity

=I derives a self-identity:

$$\begin{array}{|l} \boldsymbol{n} = \boldsymbol{n} \quad \text{=I} \end{array}$$

=E derives a sentence from another by swapping identicals:

$$\begin{array}{ll|l} m & & \mathbf{A}(\ldots \boldsymbol{n} \ldots) \\ n & & \boldsymbol{n} = \boldsymbol{m} \\ & & \mathbf{A}(\ldots \boldsymbol{m} \ldots) \quad \text{=E } m, n \end{array}$$

We may illustrate the two rules by chaining them together:

$$\begin{array}{ll|l} 1 & & a = b \\ \hline 2 & & a = a \quad \text{=I} \\ 3 & & b = a \quad \text{=E } 1, 2 \\ 4 & & a = b \rightarrow b = a \quad \rightarrow\text{I } 1\text{–}3 \end{array}$$

This shows that $\vdash \{a = b \to b = a\}$.

The previous example used **=E** on identities. But the rule can also be used on sentences that are not identities.

$$\begin{array}{l|ll} 1 & \neg((F(t,t) \leftrightarrow F(t,t)) & \\ 2 & s = t & \\ \hline 3 & \neg((F(t,t) \leftrightarrow F(t,s)) & \text{=E } 1, 2 \end{array}$$

This shows that $\{\neg((F(t,t) \leftrightarrow F(t,t)), s = t\} \vdash \neg((F(t,t) \leftrightarrow F(t,s))$.

We may also use **=E** to swap *any or all* identicals:

$$\begin{array}{l|ll} 1 & \neg((F(t,t) \leftrightarrow F(t,t)) & \\ 2 & s = t & \\ \hline 3 & \neg((F(s,s) \leftrightarrow F(s,s)) & \text{=E } 1, 2 \end{array}$$

This shows that $\{\neg((F(t,t) \leftrightarrow F(t,t)), s = t\} \vdash \neg((F(s,s) \leftrightarrow F(s,s))$.

12.2 Derivational Concepts

The derivational concepts familiar from ML carry over to FOL:

A is a THEOREM iff $\vdash$ **A**
A, **B** are INTERDERIVABLE iff **A** $\vdash$ **B** and **B** $\vdash$ **A**
Γ is INCONSISTENT iff $\Gamma \vdash \bot$; else Γ is CONSISTENT
MONOTONICTY. If $\Gamma \vdash$ **C**, then **A**, $\Gamma \vdash$ **C** (for any **A**)

Finally, FOL is both sound and complete, just as ML was:

SOUNDNESS. If $\Gamma \vdash$ **C**, then $\Gamma \vDash$ **C**
COMPLETENESS. If $\Gamma \vDash$ **C**, then $\Gamma \vdash$ **C**

CHAPTER 13
Applications

This chapter focuses on applying FOL to natural languages.

13.1 Truth-conditional Semantics

We reinterpret FOL's semantics just as we did for ML. Semantic values for *sentences* (not open formulas) are interpreted as *truth-values*.

> The SEMANTIC VALUE $[\![\cdot]\!]_{\mathbb{M}}$ for sentence $\mathbf{A}$ and model $\mathbb{M}$ is: $[\![\mathbf{A}]\!]_{\mathbb{M}} = \mathrm{T}$ iff $[\![\mathbf{A}]\!]^{\alpha}_{\mathbb{M}} = 1$ for some assignment α; else $[\![\mathbf{A}]\!]_{\mathbb{M}} = \mathrm{F}$.

13.1.1 Semantics Concepts

Reinterpreting the semantics is to reinterpret **entailment**, as in ML:

> Γ ENTAILS $\mathbf{C}$ iff there is no model $\mathbb{M}$ where $\min([\![\Gamma]\!]_{\mathbb{M}}) > [\![\mathbf{C}]\!]$

The other concepts, and their semantic tests, remain unchanged.

13.1.2 Derivations

Reinterpreting the semantics does not affect derivations.

13.2 Renderings

Strategies for rendering in FOL are much like those for ML. But FOL's many-place predicates and nested quantifiers open up new possibilities–and complexities–for rendering. To illustrate, consider:

Mike punched Bob.

Different predicates result from which names we delete. This key illustrates deleting zero (P^0), one (P^1_1, P^1_2), or both (P^2) names:

$\mathbb{D}$: people
m: Mike
b: Bob
$P^0(\cdot)$: Mike punched Bob
$P^1_1(\_1)$: ___1 punched Bob
$P^1_2(\_1)$: Mike punched Bob ___1
$P^2(\_1, \_2)$: ___1 punched ___2

This key allows us to render the original sentence as any of:

$$\begin{aligned} P^0 &\Longleftarrow \text{Mike punched Bob.} \\ P^1_1(m) &\Longleftarrow \text{Mike punched Bob.} \\ P^1_2(b) &\Longleftarrow \text{Mike punched Bob.} \\ P^2(m,b) &\Longleftarrow \text{Mike punched Bob.} \end{aligned}$$

The same sentence is rendered as *different* FOL sentences. They differ *syntactically* and *semantically*. Each has its place. But it is generally best to prefer higher-degree predicates than lower-degree predicates. This allows capturing more similarities in form. Compare:

Bob punched Mike. Jill punched Joe.

They differ over *who* punches *whom*. But both ascribe the same *relationship*: one *punching* another. The 2-place predicate P^2 captures this best because it is less specific than the lower-degree predicates.

13.2.1 Subject and Predicate

The *logical* name/predicate distinction only roughly matches the *grammatical* subject/predicate distinction. Compare:

Mike punched Bob. Bob was punched by Mike.

These sentences swap grammatical subjects ('Mike','Bob') with direct objects ('Bob','Mike'), with a shift from active ('punched') to passive ('was punched by') voice. Grammatical details like these can affect how sentences sound. If we followed them, our key would need *two* predicates: one active ('punched'), one passive ('was punched by'). But the active/passive distinction does not affect truth-conditions: Mike punched Bob if and only if Bob was punched by Mike. So, the two sentences should be rendered alike:

$\mathbb{D}$: people
m: Mike
b: Bob
$P(\_1, \_2)$: $\_\_\__1$ punched $\_\_\__2$

The relation *punched* is 2-place: one for the puncher, one for the punched. We should render it as a 2-place predicate. We may signal that P is 2-place by using a superscript: P^2. Or we may just display its slots: $P(\_1, \_2)$. Slot $\_1$ is for the puncher while slot $\_2$ is for the punched. We render the sentences as:

$P(m,b)$ $\Longleftarrow$ Mike punched Bob.
$P(m,b)$ $\Longleftarrow$ Bob was punched by Mike.

Order can matter. There's a big difference between whether Mike punched Bob and Bob punched Mike. We may capture this by plugging in different names to different slots:

$P(m,b)$ $\Longleftarrow$ Mike punched Bob.
$P(b,m)$ $\Longleftarrow$ Bob punched Mike.

The first rendering plugged m into slot $\_1$ to say that he's the puncher, while the second plugged m into slot $\_2$ to say that he's punched.

Order doesn't always matter. Consider a *symmetric* relation, such as *being near*. If one person is near another, then the second is near the first. Consider:

$\mathbb{D}$: people
a: Ali
e: Ed
$N(\_1, \_2)$: $\_\_\__1$ is near $\_\_\__2$

We may use this key to render sentences like these:

$N(a,e)$ $\Longleftarrow$ Ali is near Ed.
$N(e,a)$ $\Longleftarrow$ Ed is near Ali.

These differ *syntactically* in the order a and e are put in slot $\_1$ and slot $\_2$. For a symmetric relation, the renderings may be equivalent. But not in general for other relations.

13.2.2 Implicit Parameters

Not all slots are explicit. *Implicit parameters* illustrate this:

The rock weighs 5 pounds.

To render this, one might start with this key:

$\mathbb{D}$: everything
r: the rock
f: 5
$W(\_1, \_2)$: $\_\_\__1$ weighs $\_\_\__2$ pounds

Then the sentence could be rendered as:

$$W(r, f)$$

But changes in gravity change an object's weight. The rock would weigh 12 pounds on Jupiter, less than a pound on Earth's moon, and nothing in space's vacuum. We may have meant all along to state the rock's weight on Earth. But our rendering conceals this. Rendering 'weighs' as 2-place leaves no way to say what weight is relative to. But rendering 'weighs' as 3-place gives a way:

$\mathbb{D}$: everything
r: the rock
f: 5
e: Earth
$W(\_1, \_2, \_3)$: $\_\_\__1$ weighs $\_\_\__2$ pounds on $\_\_\__3$

We may then render the weight *relative to Earth* explicitly as:

$$W(r, f, e)$$

And we can relatvize to Jupiter or Earth's moon by naming them.

13.2.3 Nesting

Nested quantifiers open new possibilities for rendering. Consider

$\mathbb{D}$: everything
$L(\_1, \_2)$: $\_\_\__1$ loves $\_\_\__2$

We may then render various claims as:

$$\begin{array}{rcl}
\exists x \exists y L(x,y) & \Longleftarrow & \text{Someone loves someone.} \\
\exists x \forall y L(x,y) & \Longleftarrow & \text{Someone loves everyone.} \\
\forall x \exists y L(x,y) & \Longleftarrow & \text{Everyone loves someone.} \\
\forall x \forall y L(x,y) & \Longleftarrow & \text{Everyone loves everyone.}
\end{array}$$

Variables vary independently. This fits various ways to satisfy our renderings. For instance, one way for someone to love someone is for them to like *themselves*. Another way is for someone to love *someone else*. The rendering $\exists x \exists y L(x,y)$ allows both. If we want to exclude the first, we use = to express that the lover differs from the loved. These renderings illustrate how to do this in a range of cases:

$$\begin{array}{rcl}
\exists x \exists y (L(x,y) \wedge x \neq y) & \Longleftarrow & \text{Someone loves someone else.} \\
\exists x \forall y (x \neq y \rightarrow L(x,y)) & \Longleftarrow & \text{Someone loves everyone else.} \\
\forall x \exists y (L(x,y) \wedge x \neq y) & \Longleftarrow & \text{Everyone loves someone else.} \\
\forall x \forall y (x \neq y \rightarrow L(x,y)) & \Longleftarrow & \text{Everyone loves everyone else.}
\end{array}$$

13.2.4 Quantities

More quantitative statements can be rendered in FOL than before.

13.2.4.1 At least

Our strategy for rendering **indefinite descriptions** (9.2.4) generalizes to *at least* claims. Consider:

$\mathbb{D}$: everything
$B(\_1)$: $\_\_\__1$ barks

We may render 'Something barks' as:

$$\exists x B(x) \quad \Longleftarrow \quad \text{At least one thing barks.}$$

Suppose we want to say that at least *two* things bark. We may render this by saying that *two different* things bark:

$$\exists x \exists y (B(x) \wedge B(y) \wedge x \neq y) \quad \Longleftarrow \quad \text{At least two things bark.}$$

The nonidentity conjunct is essential. Without it, we have:

$$\exists x \exists y (B(x) \wedge B(y))$$

Because x, y may vary independently, they may be assigned the same value. This rendering would be true if one thing barked. So, it does *not* correctly render 'At least two things bark'.

Our strategy can be extended. We may render 'At least three things bark' as:

$$\exists x \exists y \exists z (B(x) \wedge B(y) \wedge B(z) \wedge x \neq y \wedge y \neq z \wedge x \neq z)$$
$$\Longleftarrow \text{ At least three things bark.}$$

In general, to render 'At least n things ...', we must express that at least n *distinct* things ... (for finite n).

13.2.4.2 At most

We may render *at most* claims. Reusing the key above, one rendering of 'At most one thing barks' is:

$$\neg \exists x \exists y (B(x) \wedge B(y) \wedge x \neq y) \quad \Longleftarrow \quad \text{At most one thing barks.}$$

This emphasizes that no *two* things bark. Another rendering emphasizes any "second" barking thing is just the first:

$$\forall x \forall y ((B(x) \wedge B(y)) \rightarrow x = y) \quad \Longleftarrow \quad \text{At most one thing barks.}$$

The renderings are equivalent. Neither requires that anything barks. But both set an upper limit on barking things: one.

Our strategy can be extended. We may render 'At most two things bark' in either way:

$$\neg \exists x \exists y \exists z (B(x) \wedge B(y) \wedge B(z) \wedge x \neq y \wedge y \neq z \wedge x \neq z)$$
$$\Longleftarrow \text{ At most two things barks.}$$
$$\forall x \forall y \forall z ((B(x) \wedge B(y) \wedge B(z)) \rightarrow (x = y \vee y = z \vee x = z))$$
$$\Longleftarrow \text{ At most two things bark.}$$

In general, to render 'At most n things ...', we must express that no more than n *distinct* things ... (for finite n).

13.2.4.3 Exactly

We may combine the preceding strategies: *at least* gives a lower bound, *at most* gives an upper bound. To express *exactly*, we make

the lower and upper bounds match. Let's reuse the key from before to render 'Exactly one thing barks' as:

$$\exists x(B(x) \wedge \forall y(B(y) \rightarrow y = x)) \quad \Longleftarrow \quad \text{Exactly one thing barks.}$$

This rendering embeds the upper bound (the *at most* claim) inside the lower bound (the *at least* claim). The result is an *exactly* claim. Because the lower and upper bounds match (both are one), the rendering expresses that exactly one thing barks.

Our strategy can be extended. We may render 'Exactly two things bark' as:

$$\exists x \exists y(B(x) \wedge B(y) \wedge \forall z(B(z) \rightarrow (z = x \vee z = y)))$$
$$\Longleftarrow \text{Exactly two things barks.}$$

To render 'Exactly n things ...', we must say *both* that *at least* n distinct things ... *and* that *at most* n distinct things ... (for finite n).

13.2.4.4 Definite Descriptions

While an **indefinite description** ambiguously says an item is as described without saying which (9.2.4), a **definite description** unambiguously says *the* item specified is as described. Contrast:

An English monarch is (or was) male.
The present English monarch is male.

The first sentence ambiguously describes Henry IV, George I, Charles III, ... The second sentence unambiguously describes Charles III.

Definite descriptions *uniquely* describe. This is to describe *exactly one* item. And so definite descriptions may be rendered as *exactly one* claims. Consider this key:

$\mathbb{D}$: everything
$E(\_1)$: $\_\_\_1$ is a present English monarch
$M(\_1)$: $\_\_\_1$ is male

Then we may render the second sentence as:

$$\exists x(E(x) \wedge \forall y(E(y) \rightarrow y = x) \wedge M(x))$$
$$\Longleftarrow \text{The present English monarch is male.}$$

This says that *at least* one item is a present English monarch, *at most*

one item is a present English monarch, and *that unique item* is male. And what this says is (at the time of writing) true.

The complex form of a definite description complicates rendering. To illustrate, suppose we want to *deny* that the present French monary is male. To do this, we first adjust our key:

$\mathbb{D}$: everything
$F(\_1)$: $\_\_\_1$ is a present French monarch
$M(\_1)$: $\_\_\_1$ is male

We render what we wish to deny, 'The present French monarch is male', using the previous sentence above as our guide:

$$\exists x(F(x) \wedge \forall y(F(y) \rightarrow y = x) \wedge M(x))$$
$$\Longleftarrow \text{The present French monarch is male.}$$

Denying this requires adding a negation. But where? The complex form of the rendering gives us options. One option is:

$$\exists x(F(x) \wedge \forall y(F(y) \rightarrow y = x) \wedge \neg M(x))$$
$$\Longleftarrow \text{The present French monarch is not male.}$$

But this delivers an unintended result. It says that there is exactly one present French monarch and that this monarch is not male. That is false. There is no present French monarch, male or not. So, this rendering does capture the denial we wished to express. A better rendering is:

$$\neg\exists x(F(x) \wedge \forall y(F(y) \rightarrow y = x) \wedge M(x))$$
$$\Longleftarrow \text{It is not the case that the present French monarch is male.}$$

This *denies* that there is exactly one present French monarch who is male. That is the denial we wished to express.

The renderings differ in **scope**. The first embeds the $\neg$ in the definite description, while the second embeds the definite description in the $\neg$. This scope distinction matters. The first says that *some unique thing* satisfies the description *is a present French monarch but is not male*. Because nothing satisfies that description, the rendering is false. But the second says that *nothing* satisfies the description *being the present French monarch and male*. Because nothing satisfies that description, the rendering is true.

13.3 Limitations Overcome

We can now explain how FOL overcomes ML's limitations (9.3.2).

While ML had no way to render identities, FOL does. So, we can now render arguments that "swap equals for equals", like:

2 is even.
$\sqrt{4} = 2$
$\therefore$ $\sqrt{4}$ is even.

We may use this key:

$\mathbb{D}$: everything
a: 2
b: $\sqrt{4}$
$E(\_1)$: $\_\_\_1$ is even

And we may then render the argument as :

$E(a)$
$a = b$
$\therefore$ $E(b)$

The conclusion is easily derivable from the premises:

1	$E(a)$	
2	$a = b$	
3	$E(b)$	=E 1, 2

So, $\{E(a), a = b\} \vdash E(b)$. By SOUNDNESS, $\{E(a), a = b\} \vDash E(b)$.

We may also render arguments that "infer nonidentity from difference", like:

2 is even.
3 is not even.
$\therefore$ $3 \neq 2$

We may use this key:

$\mathbb{D}$: everything
a: 2
c: 3
$E(\_1)$: $\_\_\_1$ is even

And we may then render the argument as:

$$
\begin{array}{l}
\quad E(a) \\
\quad \neg E(c) \\
\therefore\ a \neq c
\end{array}
$$

The conclusion is derivable from the premises:

1	$E(a)$	
2	$\neg E(c)$	
3	$\quad a = c$	
4	$\quad \neg E(a)$	=E 2, 3
5	$\quad \bot$	$\bot$I 1, 4
6	$\neg(a = c)$	¬I 3–5

So, $\{E(a), \neg E(c)\} \vdash a \neq c$. By SOUNDNESS, $\{E(a), \neg E(c)\} \vDash a \neq c$.

Unlike ML, FOL allows nesting quantifiers. This allows us to render this argument:

Mike punched Bob.
∴ Someone punched someone.

We may reuse a key from above:

$\mathbb{D}$: people
m: Mike
b: Bob
$P(\_1, \_2)$: $\_\_\_1$ punched $\_\_\_2$

And we may then render the argument as:

$$
\begin{array}{l}
\quad P(m,b) \\
\therefore\ \exists x \exists y P(x,y)
\end{array}
$$

The conclusion is derivable from the premises:

1	$P(m,b)$	
2	$\exists y P(n,y)$	$\exists$I 1
3	$\exists x \exists y P(x,y)$	$\exists$I 2

So, $\{P(m,b)\} \vdash \exists x \exists y P(x,y)$. By SOUNDNESS, $\{P(m,b)\} \vDash \exists x \exists y P(x,y)$.

Index

Michael J. Raven is Professor of Philosophy at the University of Victoria and Affiliate Professor of Philosophy at the University of Washington.