Knowledge Representation and Reasoning
Logics for Artificial Intelligence

Stuart C. Shapiro
Department of Computer Science and Engineering
and Center for Cognitive Science
201 Bell Hall
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000
shapiro@cse.buffalo.edu

September 17, 2009

Contents

List of Tables

Acknowledgments

The introduction is based, in part, on (Shapiro, 2003). The material on Logic is based, in part, on (Shapiro, 2000).

Part I

Introduction

Chapter 1

Knowledge Representation

> *Reports that say something hasn't happened are always interesting to me, because as we know, there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns — the ones we don't know we don't know.* [Donald Rumsfeld, February 2002]
>
> *We think we know what he means. But we don't know if we really know.* [John Lister, spokesman for Britain's Plain English Campaign, December 1, 2003]

Artificial Intelligence (AI) is a field of computer science and engineering concerned with the computational understanding of what is commonly called intelligent behavior, and with the creation of artifacts that exhibit such behavior (Shapiro, 1992a, p. 54).

Knowledge representation (KR) is a subarea of Artificial Intelligence concerned with understanding, designing, and implementing ways of representing information in computers so that programs can use this information

- to derive information that is implied by it,
- to converse with people in natural languages,
- to decide what to do next
- to plan future activities,
- to solve problems in areas that normally require human expertise.

Deriving information that is implied by the information already present is a form of *reasoning*. Because knowledge representation schemes are useless without the ability to reason with them, the field is usually known as "knowledge representation and reasoning" (though still generally abbreviated as KR). KR can be seen to be both necessary and sufficient for producing general intelligence. That is, KR is an AI-complete area (Shapiro, 1992a, p. 56).

Many philosophers consider knowledge to be justified true belief. Thus, if John believes that the world is flat, we would not say that John *knows* that the world is flat, because he is wrong—"the world is flat" is not true. Also, it may be that Sally believes that the first player in chess can always win, Betty believes that the second player can always win, and Mary believes that, with optimal play on both sides, chess will always end in a tie. One of them is correct, but we would still not say that any of them *knows* the answer, because their belief cannot have been justified by a complete analysis of the game. A computer system could not limit its information to knowledge in this strict sense, so it would be more accurate to say that the topic being discussed is *belief representation,* rather than knowledge representation. Nevertheless, we will continue to use "knowledge representation," because that has become accepted as the name of this subject.

The field of knowledge representation began, around 1958, with an investigation of how a computer might be able to represent and use the kind of commonsense knowledge we have when we decide that to get from our house to the airport, we should walk to our car and drive to the airport, rather than, for example drive to our car and then walk to the airport. The manifesto of KR may be taken to be

> *a program has common sense if it automatically deduces for itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows. . . In order for a program to be capable of learning something it must first be capable of being told it.* (McCarthy, 1959)

The phrases in this quote give us

Requirements for a Knowledge-Based Agent

1. *"what it already knows"*
   A knowledge base of beliefs.
2. *"it must first be capable of being told"*
   A way to put new beliefs into the knowledge base.
3. *"automatically deduces for itself a sufficiently wide class of immediate consequences"*
   A reasoning mechanism to derive new beliefs from ones already in the knowledge base.

In the 1960s and 1970s, much knowledge representation research was concerned with representing and using the kind of information we get from reading and from talking to other people; that is, the information that is often expressed in natural languages, and that underlies our ability to understand and to use natural languages. For example, we probably understand each of the sentences in the first column of Table 1.1 as shown in the second column, by adding our "background knowledge" to what the sentences explicitly say. Moreover, our understanding of English includes our being able to make the following inferences.

> *Every student studies hard. Therefore every smart student studies.*

Table 1.1: Some sentences and how we understand them.

Sentence	**How we understand it**
John likes ice cream.	John likes to eat ice cream.
Mary likes Asimov.	Mary likes to read books written by Isaac Asimov.
Bill flicked the switch. *The room was flooded with light.*	Bill moved the switch to the "on" position, which caused a light to come on, which lit up the room Bill was in.
Betty opened the blinds. *The courtyard was flooded with light.*	Betty adjusted the blinds so that she could see through the window they were in front of, after which she could see that the courtyard on the other side of the window was bright.

> *Tuesday evening, Jack either went to the movies, played bridge, or studied. Tuesday evening, Jack played bridge. Therefore, Jack neither went to the movies nor studied Tuesday evening.*

In the 1970s and 1980s, researchers became increasingly concerned with knowledge about specific domains in which human experts operate, such as medical diagnosis and the identification of chemical compounds from mass spectrometry data, and also with the other extreme—knowledge about the everyday world that everyone knows, such as the fact that when you tip over a glass of water, the water will spill on the floor.

In the 1980s and 1990s, these concerns focussed on the details of specific subdomains of everyday knowledge, such as theories of time and space, and also on the general structure of our knowledge of everyday terms, leading to the construction of large and general purpose "ontologies." For example, the Cyc Project has devoted many staff-years to the organization of a computer-useable representation of all the knowledge that is *not* contained in encyclopedias (Thus the name "Cyc," from "en**cyc**lopedia.") but is assumed to be already known by people who read them, and Lycos uses such an ontology to organize searches of the World-Wide Web.

In the twenty-first century, people are again interested in representing and using the kind of information expressed in natural language in order to make use of the vast store of information contained on the world-wide web.

Chapter 2

Need for a Representation Language

Although you can understand and reason about information expressed in natural language, it is clear that you don't do that by just remembering the sentences you read. This was demonstrated by Bransford and Franks in a series of experiments reported in 1971 (Bransford and Franks, 1971). Participants were shown a series of sentences, including these:

The sweet jelly was on the kitchen table.
The ants in the kitchen ate the jelly.
The ants ate the sweet jelly that was on the table.
The sweet jelly was on the table.
The jelly was on the table.
The ants ate the jelly.

Then they were given sentences, and asked if they had seen these very sentences before, and how confident they were about their answer. Participants were highly confident that they had seen the sentence

The ants ate the sweet jelly that was on the kitchen table.

Obviously, the participants remembered the information in the sentences, rather than the sentences themselves. The subject matter of KR is how might people, and how could a computer, represent such information so that they can reason about it, and how can that reasoning be carried out.

Chapter 3

Logic

In the late 1800s and early 1900s, various formal systems were developed by people who hoped, thereby, to turn human reasoning into a kind of calculation. From our perspective, we can now see that what these people were engaged in was research in knowledge representation. The formal systems they developed were systems of logic, a topic which has been studied since the days of Plato and Aristotle.

Logic is the study of correct reasoning. It is not a particular KR language. Thus, it is not proper to say "We are using (or not using) logic as our KR language." There are, indeed, many different systems of logic (*see* (Haack, 1978), (McCawley, 1981), and the various articles on Logic in (Shapiro, 1992a) beginning with (Rapaport, 1992)). So KR research may be regarded as the search for the right system of logic to use in AI systems.

A logic[1] consists of two parts: a language, and a method of reasoning. The logical language, in turn, has two aspects, syntax and semantics. Thus, to specify or define a particular logic, one needs to specify three things:

syntax: A set of atomic symbols, and the grammatical rules for combining them into well-formed expressions (symbol structures);

semantics: The meanings of the atomic symbols, and the procedures for determining the meanings of the well-formed expressions from the meanings of their component parts;

proof theory: A procedure that, given an initial set of well-formed expressions, generates additional well-formed expressions.

The way that a logic captures the notion of "correct reasoning" is that the semantics provides a procedure for identifying which well-formed expressions have a certain property, called for simplicity **Truth**, and the proof theory guarantees that, if started with a set of True well-formed expressions, all generated well-formed expressions will also be True. Of course, if the proof-theory starts with a set of well-formed expressions

[1] From now on, we will say "logic" or "logics" rather than "system of logic" or "systems of logic".

that are not all True, it may generate additional well-formed expressions that are also not True.

Various ways that systems of logic have been defined, the intricacies and variety of their syntax rules, and their semantic and proof-theoretic procedures, and how their semantic and proof-theoretic procedures may be mechanized, are the subjects of the rest of this book.

Part II

Propositional Logic

Chapter 4

Introduction

Propositional logics conceptualize domains at, but not below, the level of propositions.
A proposition is an expression in some language

- that is true or false,
- whose negation makes sense,
- that can be believed or not,
- whose negation can be believed or not,
- that can be put in the frame
  "I believe that it is not the case that _____."

Some examples of propositions are

- Betty is the driver of the car.
- Barack Obama is sitting down or standing up.
- If Opus is a penguin, then Opus doesn't fly.

Some examples of non-propositions are

- Barack Obama
- how to ride a bicycle
- If the fire alarm rings, leave the building.

A proposition is not the same as a sentence, though they are related. A sentence is an expression of a spoken or written language that, when written usually begins with a capital letter and ends with a period, question mark, or exclamation point. Some sentences do not contain a proposition. For example,

"Hi!"
"Why?"
"Pass the salt!"

Some sentences do not express a proposition, but contain one. For example

"Is Betty driving the car?"

contains the proposition

Betty is driving the car.

Some sentences contain more than one proposition. For example,

If Opus is a penguin, then Opus doesn't fly.

contains four propositions:

1. Opus is a penguin.
2. Opus flies.
3. Opus doesn't fly
4. If Opus is a penguin, then Opus doesn't fly.

Chapter 5

CarPool World

We will use CarPool World as a simple example domain. In CarPool World, Tom and Betty carpool to work. On any day, either Tom drives Betty or Betty drives Tom. In the former case, Tom is the driver and Betty is the passenger. In the latter case, Betty is the driver and Tom is the passenger.

The finest analysis of CarPool World in Propositional Logic is that there are six propositions:

Betty drives Tom.	*Tom drives Betty.*
Betty is the driver.	*Tom is the driver.*
Betty is the passenger.	*Tom is the passenger.*

We will return to CarPool World repeatedly throughout this book.

Chapter 6

The "Standard" Propositional Logic

6.1 Introduction

We will begin our look at propositional logics with the "standard," "classical," propositional logics. This is "logics" in the plural because it will be a class of standard propositional logics. We will give examples of particular propositional logics as we proceed.

6.2 Syntax

6.2.1 Syntax of Standard Propositional Logics

The syntactic expressions of propositional logics consist of **atomic propositions** and nonatomic, **well-formed propositions (wfps)**.

Syntax of Atomic Propositions

- Any letter of the alphabet, e.g.: P
- Any letter of the alphabet with a numeric subscript, e.g.: Q_3
- Any alphanumeric string, e.g.: *Tom is the driver*

is an atomic proposition.

Syntax of Well-Formed Propositions (WFPs)

1. Every atomic proposition is a wfp.
2. If P is a wfp, then so is $(\neg P)$.
3. If P and Q are wfps, then so are
   (a) $(P \wedge Q)$

(b) $(P \vee Q)$
(c) $(P \Rightarrow Q)$
(d) $(P \Leftrightarrow Q)$

4. Nothing else is a wfp.

We will not bother using parentheses when there is no ambiguity, in which case $\wedge$ and $\vee$ will have higher priority than $\Rightarrow$, which, in turn will have higher priority than $\Leftrightarrow$. For example, we will write $P \wedge Q \Leftrightarrow \neg P \Rightarrow Q$ instead of $((P \wedge Q) \Leftrightarrow ((\neg P) \Rightarrow Q))$.

We will allow $(P_1 \wedge \cdots \wedge P_n)$ and $(P_1 \vee \cdots \vee P_n)$ to be used, and will justify this later.

We will also allow matching square brackets, "[" and "]", to be used instead of parentheses.

Some example wfps are

$\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$
Tom is the driver $\Rightarrow$ *Betty is the passenger*
Betty drives Tom $\Leftrightarrow \neg$*Tom is the driver*

The symbols: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$ are also called **logical connectives.** Although we will be consistent in the use of these logical connectives, alternatives you should be aware of are shown in Table 6.1

Table 6.1: Alternative logical connectives

Our Symbol	Alternatives
$\neg$	$\sim$!
$\wedge$	& ·
$\vee$	|
$\Rightarrow$	$\rightarrow$ $\supset$ ->
$\Leftrightarrow$	$\leftrightarrow$ $\equiv$ <->

Exercise 6.1 *Which of the following are syntactically correct well-formed propositions?*

1. $(P \wedge Q \wedge R) \Rightarrow Q$
2. $(P \Rightarrow \neg Q) \vee (\neg Q \Rightarrow P)$
3. $BirdsFly \neg \Rightarrow PenguinsFly$
4. $TomIsHome \wedge \vee BettyIsHome$
5. $((P \wedge Q) \wedge R) \Leftrightarrow [P \wedge (Q \wedge R])$
6. $OpusIsAPenguin \Rightarrow \neg OpusFlies$
7. $isParent \Leftrightarrow isPerson \wedge hasChild$
8. $Tom\ drives\ Betty \Rightarrow Tom\ is\ driver$
9. $TomsOut \vee (TomsHome \wedge TomsStudying)$

6.2.2 Syntax of CarPool World Propositional Logic

The atomic propositions we will use in our propositional logic formalization of CarPool World are:

1. *Betty drives Tom*
2. *Tom drives Betty*
3. *Betty is the driver*
4. *Tom is the driver*
5. *Betty is the passenger*
6. *Tom is the passenger*

Example wfps in CarPool World Propositional Logic are:

Tom is the driver $\Rightarrow$ *Betty is the passenger*
Tom is the driver $\Rightarrow \neg$*Tom is the passenger*
Betty drives Tom $\Leftrightarrow$ *Betty is the driver* $\wedge$ *Tom is the passenger*

6.2.3 A Computer-Readable Syntax

The syntax we presented in § 6.2.1 is convenient for writing by hand and for printing in documents (such as this one), but not as convenient for input to computer programs. For that, many implemented KR systems use a version of the Common Logic Interchange Format (CLIF) (ISO/IEC, 2007).

An atomic proposition, must be recognizable as a single token by the program. So, instead of a multi-word atomic proposition like *Betty drives Tom*, one of the following is used:

Embedded underscores: `Betty_drives_Tom`

Embedded hyphens: `Betty-drives-Tom`

CamelCase: `BettyDrivesTom`

sulkingCamelCase: `bettyDrivesTom`

Escape brackets: `|Betty drives Tom|`

Quotation marks: `"Betty drives Tom"`

For non-atomic wfps, the CLIF forms that correspond to the wfps of § 6.2.1 are shown in Table 6.2.

Table 6.2: CLIF syntax

Print Form	CLIF Form
$\neg P$	`(not P)`
$P \wedge Q$	`(and P Q)`
$P \vee Q$	`(or P Q)`
$P \Rightarrow Q$	`(if P Q)`
$P \Leftrightarrow Q$	`(iff P Q)`
$(P_1 \wedge \cdots \wedge P_n)$	`(and P1 ...Pn)`
$(P_1 \vee \cdots \vee P_n)$	`(or P1 ...Pn)`

6.3 Semantics

To specify the semantics of a propositional logic, we must give the semantics of each atomic proposition and the rules for deriving the semantics of the wfps from their constituent propositions. There are actually two levels of semantics we must specify: **extensional semantics** and **intensional semantics**.

The **extensional semantics** (**value** or **denotation**) of the expressions of a logic are relative to a particular **interpretation**, **model**, **possible world**, or **situation**. The extensional semantics of CarPool World, for example, are relative to a particular day. The denotation of a proposition is either True or False. If P is an expression of some logic, we will use $[\![P]\!]$ to mean the denotation of P. If we need to make explicit that we mean the denotation relative to situation S, we will use $[\![P]\!]_S$.

The **intensional semantics** (or **intension**) of the expressions of a logic are independent of any specific interpretation, model, possible world, or situation, but are dependent only on the domain being formalized ("conceptualized"). If P is an expression of some logic, we will use $[P]$ to mean the intension of P. If we need to make explicit that we mean the intension relative to domain D, we will use $[P]_D$. Many logicians consider the intension of an expression to be a function from situations to denotations. For them, $[P]_D(S) = [\![P]\!]_S$. However, less formally, the intensional semantics of a wfp can be given as a statement in a previously understood language (for example, English) that allows the extensional value to be determined in any specific situation. Intensional semantics are often omitted when a logic is specified, but they shouldn't be.

Since the intensional semantics depends only on the domain, whereas the extensional semantics depends also on a particular situation, it is appropriate to give the intensional semantics of a particular system of logic first.

6.3.1 Intensional Semantics of CarPool World Propositional Logic

To specify the intensional semantics, we must list the atomic propositions, and, for each one, give an English interpretation. The intensional semantics of the atomic propositions of CarPool World are:

1. $[Betty\ drives\ Tom]$ = Betty drives Tom to work
2. $[Tom\ drives\ Betty]$ = Tom drives Betty to work

3. $[Betty\ is\ the\ driver]$ = Betty is the driver of the car
4. $[Tom\ is\ the\ driver]$ = Tom is the driver of the car
5. $[Betty\ is\ the\ passenger]$ = Betty is the passenger in the car
6. $[Tom\ is\ the\ passenger]$ = Tom is the passenger in the car

Note that each atomic proposition is a single indivisible symbol; the fact that the atomic propositions look like English sentences whose meanings are paraphrases of the intensional semantics is purely for mnemonic purposes. To emphasize this point, we **could have** given the following intensional semantics for CarPool World logic:

$[Betty\ drives\ Tom]$ = Tom drives Betty to work.
$[Tom\ drives\ Betty]$ = Betty drives Tom to work.
$[Betty\ is\ the\ driver]$ = Tom is the passenger in the car.
$[Tom\ is\ the\ driver]$ = Betty is the passenger in the car.
$[Betty\ is\ the\ passenger]$ = Tom is the driver of the car.
$[Tom\ is\ the\ passenger]$ = Betty is the driver of the car.

Or, we **could have** specified the atomic propositions to be A, B, C, D, E, and F, and given the following intensional semantics for CarPool World logic:

$[A]$ = Betty drives Tom to work.
$[B]$ = Tom drives Betty to work.
$[C]$ = Betty is the driver of the car.
$[D]$ = Tom is the driver of the car.
$[E]$ = Betty is the passenger in the car.
$[F]$ = Tom is the passenger in the car.

The moral is

- Don't omit specifying the intensional semantics.
- Don't presume what the intensional semantics are.
- Don't rely on "pretend it's English" semantics.

6.3.2 Intensional Semantics of WFPs

The semantics, whether intensional or extensional, of the wfps of a logic are given by listing the grammatical rules for forming wfps, and, for each one, giving a procedure for computing the semantics of a wfp from the semantics of its components. This is known as

Compositional semantics: The semantics of a non-atomic expression is a function of the semantics of its component parts and of the way they have been combined.

We will specify the intensional semantics of wfps of propositional logics by giving, for each syntactic rule, a sentence frame in which the intensional semantics of the component parts may be inserted:

- $[\neg P]$ = It is not the case that $[P]$.
- $[P \wedge Q]$ = $[P]$ and $[Q]$.
- $[P \vee Q]$ = Either $[P]$ or $[Q]$ or both.
- $[P \Rightarrow Q]$ = If $[P]$ then $[Q]$.
- $[P \Leftrightarrow Q]$ = $[P]$ if and only if $[Q]$.

Since we will use the symbols: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$ as the logical connectives for all the standard propositional logics we will discuss, these rules specify the intensional semantics of the wfps of all those logics, no matter what atomic propositions are chosen. Here are some examples for CarPool World:

- [*Tom is the driver* $\Rightarrow$ *Betty is the passenger*]
  = If Tom is the driver of the car then Betty is the passenger in the car
- [*Tom is the driver* $\Rightarrow \neg$*Tom is the passenger*]
  = If Tom is the driver of the car then it is not the case that Tom is the passenger in the car
- [*Betty drives Tom* $\Leftrightarrow$ *Betty is the driver* $\wedge$ *Tom is the passenger*]
  = Betty drives Tom to work if and only if Betty is the driver of the car and Tom is the passenger in the car

Exercise 6.2

Using the following atomic propositions, with the given intensional semantics

- $[Io\ is\ a\ moon\ of\ Jupiter]$ = *Io is a moon of Jupiter*
- $[Io\ is\ large]$ = *Io is large*
- $[Io\ is\ cold]$ = *Io is cold*
- $[Io\ is\ far\ from\ the\ Sun]$ = *Io is far from the Sun*

formalize the following sentences as well-formed propositions of Propositional Logic.

1. *If Io is far from the Sun, then Io is cold.*
2. *If Io is cold, Io is not large.*
3. *Io is a large cold moon of Jupiter.*
4. *Io is far from the Sun, but is not cold.*
5. *Io is far from the Sun if and only if it is large or a moon of Jupiter.*

Table 6.3: Five situations of CarPool World

	Denotation in Situation				
Proposition	1	2	3	4	5
Betty drives Tom	True	True	True	False	False
Tom drives Betty	True	True	False	True	False
Betty is the driver	True	True	True	False	False
Tom is the driver	True	False	False	True	False
Betty is the passenger	True	False	False	True	False
Tom is the passenger	True	False	True	False	False

6.3.3 Extensional Semantics of Atomic Propositions

The denotation of an atomic proposition is a truth value, True or False. Each way of assigning a truth value to each atomic proposition forms one situation. For example, each column of Table 6.3 gives one situation of CarPool World. Table 6.3 shows 5 situations. Since there are 6 propositions, and each one can have either of 2 truth values, there are a total of $2^6 = 64$ different situations in CarPool World. We will see in §6.3.10 how to limit these to the two that "make sense."

6.3.4 Extensional Semantics of WFPs

Just as there is a compositional intensional semantics for the wfps of propositional logics, there is a compositional extensional semantics. The standard way to compute the denotations of wfps from their constituents is:

- $[\![\neg P]\!]$ is True if $[\![P]\!]$ is False. Otherwise, it is False.
- $[\![P \wedge Q]\!]$ is True if $[\![P]\!]$ is True and $[\![Q]\!]$ is True. Otherwise, it is False.
- $[\![P \vee Q]\!]$ is False if $[\![P]\!]$ is False and $[\![Q]\!]$ is False. Otherwise, it is True.
- $[\![P \Rightarrow Q]\!]$ is False if $[\![P]\!]$ is True and $[\![Q]\!]$ is False. Otherwise, it is True.
- $[\![P \Leftrightarrow Q]\!]$ is True if $[\![P]\!]$ and $[\![Q]\!]$ are both True, or both False. Otherwise, it is False.

These can also be shown in the **truth tables** of Tables 6.4 and 6.5.

Table 6.4: Truth table defining $\neg$

P	True	False
$\neg P$	False	True

Notice that each column of these tables represents a different situation. These truth tables are the transpose of what is usually presented. Usually, each proposition heads a column and each row represents a different situation. We do it this way so that the propositions can be rather long without taking up too much space on the page.

These semantics justify the following terminology:

Table 6.5: Truth table defining $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$

P	True	True	False	False
Q	True	False	True	False
$P \wedge Q$	True	False	False	False
$P \vee Q$	True	True	True	False
$P \Rightarrow Q$	True	False	True	True
$P \Leftrightarrow Q$	True	False	False	True

- $\neg P$ is called the **negation** of P.
- $P \wedge Q$ is called the **conjunction** of P and Q. P and Q are referred to as **conjuncts**.
- $P \vee Q$ is called the **disjunction** of P and Q. P and Q are referred to as **disjuncts**.
- $P \Rightarrow Q$ is called a **conditional** or **implication**. P is referred to as the **antecedent**; Q as the **consequent**.
- $P \Leftrightarrow Q$ is called a **biconditional** or **equivalence**.

The version of implication presented here is called **material implication**. If $\Rightarrow$ is material implication, then $P \Rightarrow Q$ is True whenever P is False, regardless of whether Q is True or False. If "if ... then" is interpreted as material implication, then an "if ... then" proposition is True whenever the antecedent is False. For example, if "if ... then" is material implication, "*If the world is flat, then the moon is made of green cheese*" is True because "*The world is flat*" is False. Some logicians do not think that material implication is the correct way to formalize the English "if ... then," but that's the way it's done in standard, classical propositional logic, so we will abide by that (until later in this book).

In every situation, $P \Rightarrow Q$ and $\neg P \vee Q$ have the same denotation (are equivalent). This can be seen by examining the last two rows of Table 6.6: In fact, in some versions

Table 6.6: The equivalence of $P \Rightarrow Q$ and $\neg P \vee Q$

P	True	True	False	False
Q	True	False	True	False
$\neg P$	False	False	True	True
$P \Rightarrow Q$	True	False	True	True
$\neg P \vee Q$	True	False	True	True

of propositional logic, $P \Rightarrow Q$ is considered to be just a syntactic abbreviation of $\neg P \vee Q$. We will see this equivalence a lot.

6.3.5 Example Extensional Semantics of WFPs in CarPool World

In Table 6.7, we extend the previous table of five of the CarPool World situations by showing the denotations of two non-atomic wfps.

Table 6.7: Denotations of two CarPool World non-atomic wfps in five situations

	Denotation in Situation				
Proposition	1	2	3	4	5
Betty drives Tom	True	True	True	False	False
Tom drives Betty	True	True	False	True	False
Betty is the driver	True	True	True	False	False
Tom is the driver	True	False	False	True	False
Betty is the passenger	True	False	False	True	False
Tom is the passenger	True	False	True	False	False
$\neg$ *Tom is the driver*	False	True	True	False	True
Betty drives Tom $\Leftrightarrow \neg$ *Tom is the driver*	False	True	True	True	False

6.3.6 Uninterpreted Languages and Formal Logic

We have already slipped into a habit that is both a strength of logic (as the study of correct reasoning), and a potential confusion for the novice. Consider again Table 6.6. We said that this table shows that $P \Rightarrow Q$ is equivalent to $\neg P \vee Q$ in the sense that the two always have the same truth values in the same situations. But what happened to the rule that one should always specify the intensional semantics? By not giving the intensional semantics of P and Q, we seem to be using an "uninterpreted" logical language.

In fact, we are really making a point about **every** logical language in the class of standard classical propositional logics. Perform this thought experiment: replace all occurrences of P in the above table with any wfp of any specific standard classical propositional logic (complete with intensional semantics); replace all occurrences of Q in the table with any wfp of the same specific propositional logic (It could be the same wfp as you used for P or a different wfp.); consider the bottom two rows. In every colum, the bottom two rows will be the same. The observation does not depend on the specific logic, but only on the **form** of the wfps (and the assumption that $\Rightarrow$ is material implication). The fact that the techniques and results we are studying depend only on the form of the logical expressions, and not on their content (intensional semantics), is why this topic is often called **Formal** Logic.

When we use "uninterpreted" expressions in our study, it is not that what we are dealing with is not applicable to any practical situation, but that it is applicable to **every** practical situation formalized in a logic in the class of logics we are studying.

Exercise 6.3 *Draw truth tables for the following wfps.*

1. $P \wedge \neg P \Rightarrow Q$
2. $(P \Rightarrow Q) \vee (Q \Rightarrow P)$
3. $(P \Rightarrow Q) \Leftrightarrow (\neg P \Rightarrow \neg Q)$
4. $(P \Rightarrow Q) \Leftrightarrow (\neg Q \Rightarrow \neg P)$

6.3.7 Properties of Logical Connectives

On page 18, we said that "we will allow $(P_1 \wedge \cdots \wedge P_n)$ and $(P_1 \vee \cdots \vee P_n)$ to be used, and will justify this later." In Table 6.2, we also showed that CLIF allows the wfps, `(and P1 ...Pn)` and `(or P1 ...Pn)`. We are now ready to justify these notations.

A binary operator, $\circ$, is commutative if, for any arguments, $x, y, (x \circ y) = (y \circ x)$. That conjunction and disjunction are commutative may be seen by comparing the third column with the fourth column, and the fifth column with the sixth column in Table 6.8.

Table 6.8: Commutativity of conjunction and disjunction

A	B	$A \wedge B$	$B \wedge A$	$A \vee B$	$B \vee A$
True	True	True	True	True	True
True	False	False	False	True	True
False	True	False	False	True	True
False	False	False	False	False	False

A binary operator, $\circ$, is associative if, for any arguments, $x, y, z, ((x \circ y) \circ z) = (x \circ (y \circ z))$. That conjunction is associative may be seen by comparing the last two columns in Table 6.9, and that disjunction is associative may be seen by comparing the last two columns in Table 6.10.

Table 6.9: Associativity of conjunction

A	B	C	$A \wedge B$	$B \wedge C$	$(A \wedge B) \wedge C$	$A \wedge (B \wedge C)$
True	True	True	True	True	True	True
True	True	False	True	False	False	False
True	False	True	False	False	False	False
True	False	False	False	False	False	False
False	True	True	False	True	False	False
False	True	False	False	False	False	False
False	False	True	False	False	False	False
False	False	False	False	False	False	False

A binary operator, $\circ$, is idempotent if, for any argument, $x, x \circ x = x$. That conjunction and disjunction are idempotent may be seen by comparing the first with the second, and the first with the third columns in Table 6.11.

Consider a fully parenthesized expression all of whose operators are the same associative, commutative, idempotent, binary operator. Because the operator is associative, inner parentheses may be removed. Because the operator is commutative, the order of the operands may be permuted. Because the operator is idempotent, multiple occurrences of any one operand may be exchanged for just a single occurrence. Because conjunction and disjunction are associative, commutative, and idempotent, they may be given arbitrary numbers of arguments, with the order and multiplicity of the argu-

Table 6.10: Associativity of disjunction

A	B	C	$A \vee B$	$B \vee C$	$(A \vee B) \vee C$	$A \vee (B \vee C)$
True	True	True	True	True	True	True
True	True	False	True	True	True	True
True	False	True	True	True	True	True
True	False	False	True	False	True	True
False	True	True	False	True	True	True
False	True	False	True	True	True	True
False	False	True	False	True	True	True
False	False	False	False	False	False	False

Table 6.11: Idempotency of conjunction and disjunction

A	$A \wedge A$	$A \vee A$
True	True	True
False	False	False

ments being irrelevant. That is, they may be considered connectives that take sets of arguments.

Exercise 6.4 *Is biconditional commutative? Is it associative? Is it idempotent?*

Exercise 6.5 *Would it be correct to formalize "A, B, and C are equivalent" as $A \Leftrightarrow B \Leftrightarrow C$?*

Nor ($\downarrow$) is a logical connective with the following truth table:

A	B	$A \downarrow B$
$True$	$True$	$False$
$True$	$False$	$False$
$False$	$True$	$False$
$False$	$False$	$True$

Note that $(A \downarrow B) \Leftrightarrow \neg(A \vee B)$.

Exercise 6.6 *Is nor commutative? Is it associative? Is it idempotent?*

Nand ($|$) is a logical connective with the following truth table:

A	B	$A \mid B$
$True$	$True$	$False$
$True$	$False$	$True$
$False$	$True$	$True$
$False$	$False$	$True$

Note that $(A \mid B) \Leftrightarrow \neg(A \wedge B)$.

Exercise 6.7 *Is nand commutative? Is it associative? Is it idempotent?*

6.3.8 Computing Denotations

Using Spreadsheets to Create Truth Tables

For our first computerized KR system, we will use Microsoft Excel (any other spreadsheet, for example Google Docs Spreadsheets, should do as well). Excel has the truth values `TRUE` and `FALSE`, and the logical functions `NOT`, `AND`, and `OR`. We can use `=` as the biconditional, and `OR(NOT(`p`),`q`)` for $p \Rightarrow q$.

To set up the spreadsheet as a truth table, enter the propositions along the top-most row (you will have to use rather short propositions). If you have n atomic propositions, list them across the first n cells of the top row. The next 2^n rows should give all the possible situations. Continue along the top row, entering non-atomic propositions. In each of the 2^n cells below each non-atomic proposition, enter the spreadsheet formula to compute the value of the non-atomic proposition from the values of its components. Table 6.12 is an example, showing what you should enter. The way the spreadsheet

Table 6.12: Formulas entered in a spreadsheet to compute a truth table

	A	B	C	D
1	P	Q	∼P	P=>Q
2	TRUE	TRUE	=NOT(A2)	=OR(C2,B2)
3	TRUE	FALSE	=NOT(A3)	=OR(C3,B3)
4	FALSE	TRUE	=NOT(A4)	=OR(C4,B4)
5	FALSE	FALSE	=NOT(A5)	=OR(C5,B5)

will actually appear is shown in Table 6.13.

Table 6.13: A spreadsheet showing a truth table

	A	B	C	D
1	P	Q	∼P	P=>Q
2	TRUE	TRUE	FALSE	TRUE
3	TRUE	FALSE	FALSE	FALSE
4	FALSE	TRUE	TRUE	TRUE
5	FALSE	FALSE	TRUE	TRUE

Exercise 6.8 *Complete the spreadsheet shown in Tables 6.12 and 6.13 to include $P \wedge Q$, $P \vee Q$, and $P \Leftrightarrow Q$, as well as what's shown.*

Exercise 6.9 *Create a spreadsheet showing all 64 situations of CarPool World. You need only enter the atomic propositions for now. Save this spreadsheet for later use.*

Computing the Denotation of a Wfp in a Model

A Common Lisp program to compute the denotation of a wfp in a model is a direct implementation of the procedure outlined in § 6.3.4:

```
(defun denotation (wfp model)
  "Returns the denotation of the well-formed-proposition, wfp,
       given the model."
  ;; Use an association list for a model.
  ;;   E.g. ((P . True) (Q . False))

  ;; Use CLIF for non-atomic wfps:
  ;;     (not P)
  ;;     (and P Q)
  ;;     (or P Q)
  ;;     (if P Q)
  ;;     (iff P Q)
  (if (atom wfp)
      (cdr (assoc wfp model))
    (ecase (first wfp)
      (not (ecase (denotation (second wfp) model)
             (True 'False)
             (False 'True)))
      (and (if (and (eq (denotation (second wfp) model) 'True)
                    (eq (denotation (third wfp) model) 'True))
               'True
             'False))
      (or (if (and (eq (denotation (second wfp) model) 'False)
                   (eq (denotation (third wfp) model) 'False))
              'False
            'True))
      (if (if (and (eq (denotation (second wfp) model) 'True)
                   (eq (denotation (third wfp) model) 'False))
              'False
            'True))
      (iff (if (eq (denotation (second wfp) model)
                   (denotation (third wfp) model))
               'True
             'False)))))
```

For example,

```
cl-user(1): (denotation '(if p (if q p)) '((p . True) (q . False)))
True
cl-user(2): (denotation
             '(if BettyDrivesTom (not TomIsThePassenger))
             '((BettyDrivesTom . True) (TomIsThePassenger . True)))
False
```

Exercise 6.10 *Extend the* `denotation` *function to allow wfps of the form* `(and P1 ...Pn)` *and* `(or P1 ...Pn)`.

6.3.9 Model Finding

The techniques of § 6.3.8 were for finding the denotation of a wfp given a model. The converse problem, given the denotation of a wfp, find a model in which the wfp has that denotation, is actually more useful.

We say that a model **satisfies** a wfp if the wfp is True in that model. Obviously, if a wfp P is False in a model, $\mathcal{M}$, then $\mathcal{M}$ satisfies $\neg P$.

Exercise 6.11 *Prove that a wfp P is False in a model, $\mathcal{M}$, if and only if $\mathcal{M}$ satisfies $\neg P$.*

Similarly, we say that a model satisfies a set of wfps if they are all True in the model. It should also be obvious that a model, $\mathcal{M}$, satisfies the wfps $P_1, \ldots, P_n$ if and only if $\mathcal{M}$, satisfies $P_1 \wedge \ldots \wedge P_n$.

Exercise 6.12 *Prove that a model, $\mathcal{M}$, satisfies the wfps $P_1, \ldots, P_n$ if and only if $\mathcal{M}$, satisfies $P_1 \wedge \ldots \wedge P_n$.*

The significance of these two facts is

Observation 1 *The problem of finding a model in which a set of wfps each has a given denotation is equivalent to the problem of finding a model that satisfies a wfp.*

Model Finding with a Spreadsheet

You can use a spreadsheet to find satisfying models of a set of wfps.

1. Set up the spreadsheet according to the instructions on page 28:
   (a) put the n atomic propositions in the first n columns of the top row;
   (b) enter truth values in the next 2^n rows so that each row is one of the possible models;
   (c) enter non-atomic propositions in the first row after the atomic propositions;
   (d) enter appropriate formulas in the 2^n rows under each non-atomic proposition.
2. Use your spreadsheet's table (or list) feature to make a table of the entire truth table. Each column should be headed by a wfp, and the entries should appear as truth values.
3. For each wfp that you want to be satisfied, filter its column so that only rows in which that wfp has the value of True appear.
4. Each visible row represents a satisfying model of the wfps, specified by the truth values of the atomic wfps that head the first n columns.

Exercise 6.13 *Extend the spreadsheet truth table for CarPool World that you created for Exercise 6.9, and use it to find the satisfying models for $Betty\ is\ the\ driver \wedge \neg\, Tom\ is\ the\ driver$. How many satisfying models are there? What are they?*

The Tableau Procedure for Model Finding

We will give two versions of a procedure for finding satisfying models that is easy to do by hand. It is variously called the method of **tableaux** or **semantic tableaux** or **analytic tableaux**, and is due to (Beth, 1959), (Wang, 1960), and (Smullyan, 1968). We will call it the tableau procedure. The first version is a sketch that should be easy to understand. The second version organizes this sketch into a formal algorithm.

Both versions begin with a set of wfps for which we want to find a satisfying model. They label each of the wfps True, and follow the procedure of § 6.3.4 backward recursively. The first, less formal version is:

- Given: Wfps labeled True, False, or unlabeled.
- If any wfp is labeled both True and False, terminate with failure.
- If all the atomic wfps are labeled, return the labeling as a model.
- If $\neg P$ is
  - labeled True, try labeling P False.
  - labeled False, try labeling P True.
- If $P \wedge Q$ is
  - labeled True, try labeling P and Q True.
  - labeled False, try labeling P False, and try labeling Q False.
- If $P \vee Q$ is
  - labeled False, try labeling P and Q False.
  - labeled True, try labeling P True, and try labeling Q True.
- If $P \Rightarrow Q$ is
  - labeled False, try labeling P True and Q False.
  - labeled True, try labeling P False, and try labeling Q True.
- If $P \Leftrightarrow Q$ is
  - labeled True, try labeling P and Q both True, and try labeling P and Q both False.
  - labeled False, try labeling P True and Q False, and try labeling P False and Q True.

Notice that each sub-bullet considers a case where a non-atomic wfp has a given label and "tries" either one or two labellings for the components of the non-atomic wfp. For example, if $P \wedge Q$ is labeled True, that can only be because P and Q are both True. That is, the only models that satisfy $P \wedge Q$ are models in which P and Q are both True. On the other hand, if $P \wedge Q$ is labeled False, that could be either because P is labeled False or because Q is labeled False. That is, the only models that satisfy $\neg(P \wedge Q)$

are models in which P is False (and Q could be either True or False) or in which Q is False (and P could be either True or False).

The second version of the tableau procedure involves drawing a tree diagram. The root of the tree will show all the wfps labeled "T", for True. Each step of the procedure will consist of either:

- "closing" a branch of the tree because it contains some wfp labeled both "T" and "F";
- extending one leaf of the tree by choosing one non-atomic wfp in its branch that has not yet been "reduced" in this branch, and giving this leaf one or more child nodes.
- marking a branch as finished because there are no more non-atomic wfps in its branch that have not yet been reduced in this branch.

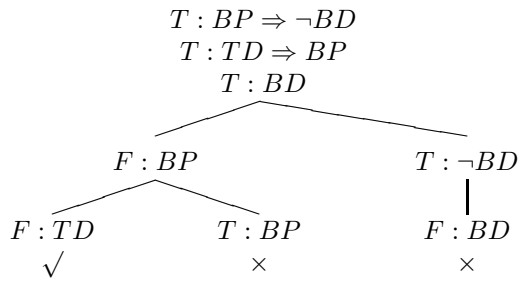
When the procedure terminates, every finished branch gives a satisfying model of the original set of wfps. The procedure is:

1. Draw the root of the tree with each wfp in the original set of wfps labeled "T".
2. While there is a leaf node with neither a check mark ("$\surd$") nor an x mark ("$\times$") under it, choose one such leaf node and do the following:
   - If any wfp in the branch ending at the leaf node is labeled "T" in some node and "F" in some node, close the branch by writing "$\times$" below the leaf.
   - Else, if there is no non-atomic wfp that is written in some node in the branch ending at the leaf node, that has not yet been "reduced" in this branch, indicate that the branch is finished by writing "$\surd$" below the leaf.
   - Else choose one non-atomic wfp that is written in some node in the branch ending at the leaf node, but has not yet been "reduced" in this branch, and reduce it by doing the following:
     - If the non-atomic wfp is of the form $\neg P$ and is
       - labeled "T", add a new node below the chosen leaf, and write in it "$F : P$".
       - labeled "F", add a new node below the chosen leaf, and write in it "$T : P$".
     - If the non-atomic wfp is of the form $P_1 \wedge \ldots \wedge P_n$ and is
       - labeled "T", add a new node below the chosen leaf, and write in it "$T : P_1$" and ... and "$T : P_n$".
       - labeled "F", add n new nodes below the chosen leaf, and write in the i^{th} node "$F : P_i$".
     - If the non-atomic wfp is of the form $P_1 \vee \ldots \vee P_n$ and is
       - labeled "T", add n new nodes below the chosen leaf, and write in the i^{th} node "$T : P_i$".
       - labeled "F", add a new node below the chosen leaf, and write in it "$F : P_1$" and ... and "$F : P_n$".

- If the non-atomic wfp is of the form $P \Rightarrow Q$ and is
  - labeled "T", add two new nodes below the chosen leaf, and write in one "$F : P$" and in the other "$T : Q$".
  - labeled "F", add a new node below the chosen leaf, and write in it "$T : P$" and "$F : Q$".
- If the non-atomic wfp is of the form $P \Leftrightarrow Q$ and is
  - labeled "T", add two new nodes below the chosen leaf, and write in one both "$T : P$" and "$T : Q$", and in the other both "$F : P$" and "$F : Q$".
  - labeled "F", add two new nodes below the chosen leaf, and write in one both "$T : P$" and "$F : Q$", and in the other both "$F : P$" and "$T : Q$".

3. If any branch ends with a "$\surd$", the labellings of the atomic wfps on that branch give a satisfying model of the original set of wfps. If any atomic wfp in the original set is not labeled in this branch, then it could be either True or False in this satisfying model.

For example, using the tableau procedure to find satisfying models of the set of wfps, $\{BP \Rightarrow \neg BD, TD \Rightarrow BP, BD\}$ produces the tree:



Under the root node, we reduced $BP \Rightarrow \neg BD$. Under the $F : BP$ node, we reduced $TD \Rightarrow BP$. Under the $T : \neg BD$ node, we reduced $\neg BD$. The right-most branch contains both $T : BD$ and $F : BD$. The middle branch contains both $F : BP$ and $T : BP$. The left-most branch shows that $[\![BD]\!]$ = True, $[\![BP]\!]$ = False, and $[\![TD]\!]$ = False is a satisfying model.

We will call a tree drawn by following the tableau procedure a tableau.

Exercise 6.14 *Prove that the tableau procedure terminates for any finite original set of wfps.*

Exercise 6.15 *Prove that the tableau procedure is correct. That is, any branch that gets a "$\surd$" written at its end provides a satisfying model of the original set of wfps.*

Exercise 6.16 *Prove that the tableau procedure is complete. That is, for any finite original set of wfps, and for any satisfying model of the set, this procedure will find the satisfying model.*

Exercise 6.17 *Prove that the tableau procedure is, in the worst case, exponential in the number of atomic wfps in the original set of wfps. That is, if there are n different atomic wfps in the original set of wfps, this procedure may produce a tree with 2^n branches.*

Exercise 6.18 *Draw tableaux to find satisfying models of each of the following set of wfps, and for each set, indicate the satisfying model found in your tableau.*

1. $\{(\neg M \Rightarrow (S \vee P)), ((S \Rightarrow H) \wedge (P \Rightarrow K)), \neg K\}$
2. $\{(A \Rightarrow B \vee C), (P \Rightarrow Q \wedge R), \neg(C \vee R)\}$
3. $\{((A \Rightarrow B) \Leftrightarrow (P \wedge Q)), (\neg P \Rightarrow A), \neg P\}$

An Implementation of the Tableau Procedure

Following is a recursive Common Lisp program that finds satisfying models using the tableau procedure. The program, `models`, takes a list of wfps to be satisfied, and, optionally, up to three more lists: a list of wfps that should all be False in the models; a list of atomic wfps that should be True in the models; and a list of atomic wfps that should be False in the models. At each recursive loop, `models`, removes one wfp from either of the first two lists, and either puts it in one of the lists of atomic wfps, or puts its components in the appropriate lists of possibly non-atomic wfps. If there is ever a non-empty intersection of the first two lists or of the third and fourth list, the empty model is returned. If the first two lists are both empty, then the model assigns True to the atomic wfps in the third list and False to the atomic wfps in the fourth list. In approximately half of the recursive cases, `models` calls itself twice, in which case it appends the two lists of satisfying models.

The `models` program uses two "helper" functions:

```
(defun add (wfp wfps)
  "If wfp is already on the list wfps, returns wfps;
     otherwise, returns a copy of wfps with wfp added to it."
  (adjoin wfp wfps :test 'equal))

(defun addall (newwfps oldwfps)
  "Returns a list of all the wfps on either newwfps or oldwfps."
  (union newwfps oldwfps :test 'equal))
```

Here is the `models` program itself:

```
(defun models (trueWfps &optional falseWfps trueAtoms falseAtoms)
  "Returns the set of models that:
       satisfy the list wfps in trueWfps;
       falsify the wfps in falseWfps;
       satisfy the list of atomic wfps in trueAtoms;
       and falsify the list of atomic wfps in falseAtoms.
    If there are no such models, returns nil."
```

```
;; Use an association list for a model.
;;   E.g. ((P . True) (Q . False))

;; Use CLIF for non-atomic wfps:
;;     (not P)
;;     (and P1 ... Pn)
;;     (or P1 ... Pn)
;;     (if P Q)
;;     (iff P Q)
(cond
 ((or (intersection trueAtoms falseAtoms)
      (intersection trueWfps falseWfps :test 'equal))
  nil)
 (trueWfps
  (let ((wfp (first trueWfps)))
    (cond ((atom wfp)
           (models (rest trueWfps) falseWfps
                   (add wfp trueAtoms) falseAtoms))
          (t (case (first wfp)
               (not (models (rest trueWfps)
                            (add (second wfp) falseWfps)
                            trueAtoms falseAtoms))
               (and (models (addall (rest wfp) (rest trueWfps))
                            falseWfps trueAtoms falseAtoms))
               (or (loop for disjunct in (rest wfp)
                         append (models
                                  (add disjunct (rest trueWfps))
                                  falseWfps trueAtoms falseAtoms)))
               (if (append (models (add (third wfp) (rest trueWfps))
                                   falseWfps trueAtoms falseAtoms)
                           (models (rest trueWfps)
                                   (add (second wfp) falseWfps)
                                   trueAtoms falseAtoms)))
               (iff (append
                      (models (add (second wfp)
                                   (add (third wfp) (rest trueWfps)))
                              falseWfps trueAtoms falseAtoms)
                      (models (rest trueWfps)
                              (add (second wfp)
                                   (add (third wfp) falseWfps))
                              trueAtoms falseAtoms)))))))))
 (falseWfps
  (let ((wfp (first falseWfps)))
    (cond ((atom wfp)
           (models trueWfps (rest falseWfps)
```

```
                    trueAtoms (add wfp falseAtoms)))
           (t (case (first wfp)
                (not (models (add (second wfp) trueWfps)
                             (rest falseWfps)
                             trueAtoms falseAtoms))
                (and (loop for conjunct in (rest wfp)
                           append (models
                                   trueWfps
                                   (add conjunct (rest falseWfps))
                                   trueAtoms falseAtoms)))
                (or (models trueWfps
                            (addall (rest wfp) (rest falseWfps))
                            trueAtoms falseAtoms))
                (if (models (add (second wfp) trueWfps)
                            (add (third wfp) (rest falseWfps))
                            trueAtoms falseAtoms))
                (iff (append
                      (models (add (second wfp) trueWfps)
                              (add (third wfp) (rest falseWfps))
                              trueAtoms falseAtoms)
                      (models (add (third wfp) trueWfps)
                              (add (second wfp) (rest falseWfps))
                              trueAtoms falseAtoms)))))))
  (t (list (append (mapcar #'(lambda (a) (cons a 'True))
                           trueAtoms)
                   (mapcar #'(lambda (a) (cons a 'False))
                           falseAtoms))))))
```

Here are some tests of `models`:

```
cl-user(1): (models '( (if BP (not BD)) (if TD BP) BD))
(((BD . True) (BP . False) (TD . False)))

cl-user(2): (models '( BDT (if BDT (and BD TP)) (not (or TP BD))))
nil

cl-user(3): (models '( (if BDT (and BD TP)) (if TDB (and TD BP))))
(((TD . True) (BP . True) (BD . True) (TP . True))
 ((BD . True) (TP . True) (TDB . False))
 ((TD . True) (BP . True) (BDT . False))
 ((BDT . False) (TDB . False)))
```

The first test of `models` returns one satisfying model; the second returns no satisfying model. The third test looks like it returns four satisfying models, but those four actually represent a lot more. There are six atomic propositions in the wfps `(if BDT (and BD TP))` and `(if TDB (and TD BP))`. The first result of `models` shows truth values for only four of them, so the remaining two, `BDT` and `TDB`, could

have any truth values. Therefore, the first result actually represents $2^2 = 4$ models. Similarly, the second result represents 8 models; the third represents 8 models, and the fourth result represents 16 models. There are not, however, 36 satisfying models of `(if BDT (and BD TP))` and `(if TDB (and TD BP))` because some of them are repeats.

Exercise 6.19 *How many satisfying models of `(if BDT (and BD TP))` and `(if TDB (and TD BP))` are there? What are they?*

Model Finding with Decreasoner

The `models` program written above was written for pedagogical purposes. It was not written to be especially efficient. A model-finding program that was written to be efficient is `decreasoner`[1] (Mueller, 2006; Mueller, 2008). We will discuss how `decreasoner` operates later in this book. For now we will demonstrate its use using the `ubdecreasonerP.py` front end, and the first example given to `models` above. First, we prepare an input file,
`/projects/shapiro/CSE563/Examples/Decreasoner/cpwExample.e`
containing

```
;;; Propositional Car Pool World Example
;;; Stuart C. Shapiro
;;; December 2, 2008

proposition BettyIsDriver ; Betty is the driver of the car.
proposition TomIsDriver ; Tom is the driver of the car.
proposition BettyIsPassenger ; Betty is the passenger in the car.

BettyIsPassenger -> !BettyIsDriver.
TomIsDriver -> BettyIsPassenger.
BettyIsDriver.
```

Then we run it from the Linux command line:[2]

```
<nickelback:~:1:53> cd /projects/shapiro/CSE563/decreasoner

<nickelback:decreasoner:1:54> python ubdecreasonerP.py
                               ../Examples/Decreasoner/cpwExample.e
Copyright (c) 2005 IBM Corporation and others.
All rights reserved. This program and the accompanying materials
are made available under the terms of the Common Public License v1.0
which accompanies this distribution, and is available at
http://www.eclipse.org/legal/cpl-v10.html

Contributors:
```

[1] available from `http://decreasoner.sourceforge.net/`

[2] The second command was split into two lines to fit on this page. It should be entered on one line.

```
IBM - Initial implementation

---

model 1:

BettyIsDriver.
!BettyIsPassenger.
!TomIsDriver.
```

Notice that the model found by `decreasoner` is the same one found by `models`.

Exercise 6.20 *Use `ubdecreasonerP.py` to find satisfying models of `(if BDT (and BD TP))` and `(if TDB (and TD BP))`. How many does it report? What are they?*

6.3.10 Semantic Properties of WFPs

We have already seen that, for a given wfp, there may be no model that satisfies it, or there might be one satisfying model, or there might be more than one satisfying model. In general, a wfp is either **satisfiable, contingent, valid,** or **contradictory** according to the situations (models) in which it is True. A wfp is

satisfiable if it is True in at least one situation;

contingent if it is True in at least one situation and False in at least one situation;

valid if it is True in every situation;

contradictory if it is False in every situation.

For example, as Table 6.14 shows, $\neg P$, $Q \Rightarrow P$, and $P \Rightarrow (Q \Rightarrow P)$ are satisfiable, $\neg P$ and $Q \Rightarrow P$ are contingent, $P \Rightarrow (Q \Rightarrow P)$ is valid, and $P \wedge \neg P$ is contradictory.

Table 6.14: Truth table showing satisfiable, contingent, valid, and contradictory wfps

P	True	True	False	False
Q	True	False	True	False
$\neg P$	False	False	True	True
$Q \Rightarrow P$	True	True	False	True
$P \Rightarrow (Q \Rightarrow P)$	True	True	True	True
$P \wedge \neg P$	False	False	False	False

If A is a well-formed proposition of a logic $\mathcal{L}$, it is standard to write $\models_{\mathcal{L}} A$ to indicate that A is valid in logic $\mathcal{L}$. The subscript may be omitted if it is clear from context. Thus, Table 6.14 table shows that $\models P \Rightarrow (Q \Rightarrow P)$. Valid wfps are also called **tautologies.** The symbol "$\models$" is called a "double turnstyle". You should read "$\models A$" as "*A is valid.*" We also write "*A is not valid*" as "$\not\models A$." Notice that if A is not valid, it still might be satisfiable.

Related to the notion of validity is the notion of **logical entailment** (sometimes called "logical implication", "semantic entailment", or "semantic implication"). The set of wfps $\{A_1, \ldots, A_n\}$ logically entails the wfp B in logic $\mathcal{L}$ (written $A_1, \ldots, A_n \models_{\mathcal{L}} B$) if and only if B is True in every situation in which every A_i is True. For example, Table 6.14 shows that $Q, Q \Rightarrow P \models P$ and $P \models Q \Rightarrow P$.

6.3.11 The KRR Enterprise

Domain knowledge is used to reduce the set of situations to only those that "make sense." For example, in CarPool World, we want to specify that:

- Betty is the driver or the passenger, but not both:

$$Betty\ is\ the\ driver \Leftrightarrow \neg Betty\ is\ the\ passenger$$

- Tom is the driver or the passenger, but not both:

$$Tom\ is\ the\ driver \Leftrightarrow \neg Tom\ is\ the\ passenger$$

- If Betty drives Tom, then Betty is the driver and Tom is the passenger:

$$Betty\ drives\ Tom \Rightarrow Betty\ is\ the\ driver \wedge Tom\ is\ the\ passenger$$

- If Tom drives Betty, then Tom is the driver and Betty is the passenger:

$$Tom\ drives\ Betty \Rightarrow Tom\ is\ the\ driver \wedge Betty\ is\ the\ passenger$$

- Finally, either Tom drives Betty or Betty drives Tom:

$$Tom\ drives\ Betty \vee Betty\ drives\ Tom$$

Table 6.15 shows the only two situations of CarPool World (numbered as in Table 6.3) in which all five of these domain rules are True. Notice that these are precisely

Table 6.15: Truth table of "common sense" situations of CarPool World

	Denotation in Situation	
Proposition	3	4
Betty drives Tom	True	False
Tom drives Betty	False	True
Betty is the driver	True	False
Tom is the driver	False	True
Betty is the passenger	False	True
Tom is the passenger	True	False

the two commonsense situations.

Exercise 6.21 *Extend the spreadsheet you created for Exercise 6.9 to include the five domain rules for CarPool World shown above. Use the technique described in § 6.3.9 so that only the situations that satisfy the domain rules appear. Convince yourself that these are the two situations shown in Table 6.15. Save this spreadsheet.*

Exercise 6.22 *Using the spreadsheet you created for Exercise 6.21, convince yourself that the English intensional semantics given above for each domain rule is correct. Do this by filtering the spreadsheet by each domain rule by itself. The situations that satisfy each domain rule should be the ones described by the English intensional semantics.*

The number of situations that satisfy a set of wfps is monotonic nondecreasing as the set of wfps is increased. In general, as we add domain rules, the number of situations that satisfy them will be reduced, although it might be the case that when we add a new domain rule, the number of satisfying situations will remain the same. In that case, the last domain rule added is not **independent** of the previous domain rules, and there is no reason to include it as a domain rule of the particular domain. The best possible formalization of a domain is achieved when the smallest possible set of independent domain rules that reduce the number of satisfying models to those that "make sense" is found. There may, however, be several equally small independent sets.

Exercise 6.23 *Using the spreadsheet you created for Exercise 6.21, convince yourself that the five CarPool World domain rules given above are independent. Do this by starting with the full spreadsheet, then filter it by each domain rule one at a time, then two at a time, etc. It should be the case that, whichever domain rules are "active," when you add one more, the number of satisfying situations decreases.*

If we let Γ stand for the set of CarPool World domain rules, and A be any wfp in the logic of CarPool World, then $\Gamma \models A$ says that the domain rules logically imply A. That means that A is True in every model that satisfies the domain rules—every situation that "makes sense." This is how every KR system is used (when a Propositional Logic is used to formalize the domain):

1. Given a domain you are interested in reasoning about.
2. List the set of propositions (expressed in English) that captures the basic information of interest in the domain.
3. Formalize the domain by creating one atomic wfp for each proposition listed in step (2). List the atomic wfps, and, for each, show the English proposition as its intensional semantics.
4. Using the atomic wfps, determine a set of domain rules so that all, but only, the situations of the domain that make sense satisfy them. Strive for a set of domain rules that is small and independent.
5. Optionally, formulate an additional set of situation-specific wfps that further restrict the domain to the set of situations you are interested in. We will call this restricted domain the "subdomain".

6. Letting Γ be the set of domain rules plus situation-specific wfps, and A be any proposition you are interested in, A is True in the subdomain if $\Gamma \models A$, is false in the subdomain if $\Gamma \models \neg A$, and otherwise is True in some more specific situations of the subdomain, and False in others.

These steps are what this book is all about. Steps (2)–(5) constitute the Knowledge Representation problem. Determining if $\Gamma \models A$ constitutes the Reasoning problem. As we proceed, we will study various logics that enable Steps (2)–(4), and various computational methods to determine if $\Gamma \models A$.

The techniques you know at this point to determine if $\Gamma \models A$ is to use the techniques of §6.3.9 to find the models that satisfy Γ, and then to use the techniques of §6.3.8 to find the denotation of A in each of those models. If A is True in every model that satisfies Γ, then $\Gamma \models A$.

Exercise 6.24 *Letting Γ be the set of CarPool World domain rules, determine if*

$$\Gamma \models Tom\ is\ the\ passenger \Rightarrow Betty\ drives\ Tom$$

If we want to reason about fewer situations than all of those that satisfy the domain rules, we add them to the left-hand side of the $\Gamma \models A$ expression. For example, for some Γ, A, B, and C, we might investigate whether $\Gamma, A, B \models C$. The collection of wfps on the left-hand side can be considered a **knowledge base (KB)**. The question then is, is it the case that $KB \models C$?

Exercise 6.25 *Letting Γ be the set of CarPool World domain rules, determine if*

$$\Gamma, Betty\ drives\ Tom \models \neg Tom\ is\ the\ driver$$

In English, this problem is: In CarPool World, is it the case that on days when Betty drives Tom, Tom is not the driver?

It is easy to show that

Metatheorem 1 *$A_1, \ldots, A_n \models_{\mathcal{L}} B$ if and only if $A_1 \wedge \cdots \wedge A_n \models_{\mathcal{L}} B$*

We will, therefore, sometimes talk about a set of wfps logically implying a wfp, and sometimes talk about a wfp logically implying a wfp.

Logical implication and logical validity are related by the following

Metatheorem 2 *$A \models_{\mathcal{L}} B$ if and only if $\models_{\mathcal{L}} A \Rightarrow B$*

The significance of this is

Observation 2 *If one is interested in determining either logical validity or logical implication, one may solve the other problem, and then, citing Metatheorem 2, show the solution of the original problem.*

6.4 Computational Methods for Determining Entailment and Validity

As you have just seen, given a knowledge base KB and a query Q, one way to determine if Q is True in the domain described by KB, that is if $KB \models Q$, is to find all models that satisfy KB, and then see if Q is True in each one of them. The program `models`, shown on page 34, returns the set of models that satisfies a wfp, and the program `denotation`, shown on page 28, computes the denotation of a wfp in a model. So a first draft of a procedure to decide whether $KB \models Q$ is to loop through every model, `M`, returned by `(models KB)`, and make sure that `(denotation Q M)` returns True for each one of them. This procedure is captured by the program `entails`:

```
(defun entails (KB Q)
  "Returns t if the knowledge base KB entails the query Q;
    else returns nil."
  ;; Uses models to find the set of models that satisfy KB.
  ;; Then uses denotation to see if Q is true in every satisfying model.

  ;; Use CLIF for non-atomic wfps:
  ;;     (not P)
  ;;     (and P Q)
  ;;     (or P Q)
  ;;     (if P Q)
  ;;     (iff P Q)
  (loop for model in (models KB)
      unless (denotation Q model)
      do (return-from entails nil))
  t)
```

For example,

```
cl-user(1): (entails '( (if BP (not BD)) (if TD BP) BD )
                     '(and (not BP) (not TD)))
t
```

There are two problems with this program:

1. as we noted on page 36, `models` does not really return all the satisfying models—some of the "models" it returns represents multiple models with some atomic wfps having different denotations in different ones;
2. after `models` tries to generate all the satisfying models, `denotation` goes through every one evaluating an arbitrarily complex wfp.

Exercise 6.26 *What does `entails` do if you try*

```
(entails '( (if BDT (and BD TP)) (if TDB (and TD BP)))
         '(or (not BD) (not TD)))
```

A better idea is based on the idea that $KB \models Q$ if and only if no model satisfies $KB \wedge \neg Q$. Methods based on this idea are called **Tableau** methods or **Semantic Tableau** methods. We will also refer to them as **Model-Finding Refutation** methods.

The Wang Algorithm (Wang, 1960) is a model-finding refutation procedure that is just a slight reorganization of the `models` program. `wang` takes two lists of wfps, $Twfps$ and $Fwfps$, and "tries" to find a model that both satisfies the wfps in $Twfps$ and falsifies the wfps in $Fwfps$. However, if it succeeds in this, it returns False. If there is no such model, `wang` returns True.

```
wang(Twfps, Fwfps) {
   /*
    * Twfps and Fwfps are sets of wfps.
    * Returns True if there is no model
    * that satisfies Twfps and falsifies Fwfps;
    * Otherwise, returns False.
   */

   if Twfps and Fwfps intersect, return True;
   if every A ∈ Twfps ∪ Fwfps is atomic, return False;
   if (P = ¬A) ∈ Twfps, return wang(Twfps \ {P}, Fwfps ∪ {A});
   if (P = ¬A) ∈ Fwfps, return wang(Twfps ∪ {A}, Fwfps \ {P});
   if (P = A ∧ B) ∈ Twfps, return wang((Twfps \ {P}) ∪ {A, B}, Fwfps);
   if (P = A ∧ B) ∈ Fwfps, return wang(Twfps, (Fwfps \ {P}) ∪ {A})
                                  and wang(Twfps, (Fwfps \ {P}) ∪ {B});
   if (P = A ∨ B) ∈ Twfps, return wang((Twfps \ {P}) ∪ {A}, Fwfps);
                                  and wang((Twfps \ {P}) ∪ {B}, Fwfps);
   if (P = A ∨ B) ∈ Fwfps, return wang(Twfps, (Fwfps \ {P}) ∪ {A, B})
   if (P = (A ⇒ B)) ∈ Twfps, return wang(Twfps \ {P}, Fwfps ∪ {A})
                                    and wang((Twfps \ {P}) ∪ {B}, Fwfps);
   if (P = (A ⇒ B)) ∈ Fwfps, return wang(Twfps ∪ {A}, (Fwfps \ {P}) ∪ {B});
   if (P = (A ⇔ B)) ∈ Twfps, return wang((Twfps \ {P}) ∪ {A, B}, Fwfps)
                                    and wang(Twfps \ {P}, Fwfps ∪ {A, B});
   if (P = (A ⇔ B)) ∈ Fwfps, return wang(Twfps ∪ {A}, (Fwfps \ {P}) ∪ {B})
                                    and wang(Twfps ∪ {B}, (Fwfps \ {P}) ∪ {A});
}
```

Using `wang`, we can rewrite `entails` as:

```
(defun entails (KB Q)
  "Returns t if the knowledge base KB entails the query Q;
    else returns nil."
  ;; Uses wang to determine that no model satisfies KB
  ;;      while falsifying Q.

  ;; Use CLIF for non-atomic wfps:
  ;;     (not P)
  ;;     (and P Q)
```

```
  ;;     (or P Q)
  ;;     (if P Q)
  ;;     (iff P Q)
  (wang KB (list Q)))
```

For example,

```
cl-user(4): (entails '( (if BP (not BD)) (if TD BP) BD )
                     '(and (not BP) (not TD)))
t
```

Exercise 6.27 *What does this version of* `entails` *do if you try*

```
(entails '( (if BDT (and BD TP)) (if TDB (and TD BP)))
         '(or (not BD) (not TD)))
```

Exercise 6.28 *Formalize the following domain.*[3]

> **Background (Domain) knowledge:** *If there is a good movie on TV and Tom doesn't have an early appointment the next morning, then he stays home and watches a late movie. If Tom needs to work and doesn't have an early appointment the next morning, then he works late. If Tom works and needs his reference materials, then he works at his office. If Tom works late at his office, then he returns to his office. If Tom watches a late movie or works late, then he stays up late.*
>
> **To Assume (Situation-specific knowledge):** *Tom needs to work, doesn't have an early appointment, and needs his reference materials.*
>
> **To Prove:** *Tom returns to his office and stays up late.*

Using this version of `entails`*, show that background knowledge plus the assumptions logically entail the sentence to be proved.*

6.5 Proof Theory

There are two basic varieties of proof theory (or syntactic inference methods) in propositional logics, **Hilbert-style** methods, and **natural deduction** methods. Hilbert-style inference methods use a large number of **(logical) axioms** and a small number of **rules of inference,** whereas natural deduction methods use a small number of (logical) axioms (or even none at all) and a large number of rules of inference.[4] Usually there are two rules of inference for each **logical connective**, $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$, an **introduction rule**, and an **elimination rule**. These are usually abbreviated by writing the logical connective before "I" or "E", respectively. For example $\neg\mathbf{I}$ is the "negation introduction" rule, and $\wedge\mathbf{E}$ is the "and elimination" rule. The rule $\Rightarrow$ **E** is also often called **modus ponens**.

[3] Based on an example in (Shapiro, 1987, 779–785).

[4] According to (Pelletier, 2000) the real distinguishing feature of natural deduction methods is that they include a method of using subproofs.

6.5.1 Hilbert-Style Methods

In Hilbert-style methods, a **derivation** of a wfp, A, from a set of assumptions (or **non-logical axioms**,) Γ, is a list of wfps in which each wfp in the list is either a logical axiom, or a non-logical axiom, or follows from previous wfps in the proof according to one of the rules of inference. A Hilbert-style **proof** of a wfp, A, is a derivation of A from an empty set of assumptions. If A can be derived from Γ in the logic $\mathcal{L}$, we write $\Gamma \vdash_{\mathcal{L}} A$, (The symbol "$\vdash$" is called a "turnstyle".) while if A can be proved in $\mathcal{L}$, we write $\vdash_{\mathcal{L}} A$. If A can be proved in $\mathcal{L}$, A is called a **theorem** of $\mathcal{L}$.

One set of axioms for the standard propositional calculus is (Mendelson, 1964, p. 31):

(A1). $(A \Rightarrow (B \Rightarrow A))$

(A2). $((A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C)))$

(A3). $((\neg B \Rightarrow \neg A) \Rightarrow ((\neg B \Rightarrow A) \Rightarrow B))$

Note that these axioms use only the connectives $\Rightarrow$ and $\neg$. The others may be defined as abbreviations of these, or additional axioms may be given for them. For our purposes, we will not bother about them.

There are two attitudes taken about axioms such as these:

1. They are *axiom schemata.* The actual axioms (There are an infinite number of them.) may be derived from them by taking any one axiom schema, and substituting any wfp whatsoever for all occurrences of each of the letters, A, B, and C. We will take this attitude in the example below.

2. They are actual axioms, and the rule of Substitution, sketched above, is a Rule of Inference.

Whether (A1) – (A3) are axioms or axiom schemata, the (other) Rule of Inference is *modus ponens,* which says that if any wfp A is a line of a proof, and a wfp of the form $A \Rightarrow B$ is also a line in the proof, for any wfp B, then B can be added as a line in the proof. This rule of inference can be summarized as

$$\frac{A, A \Rightarrow B}{B}$$

An example proof that $\vdash A \Rightarrow A$ is (Mendelson, 1964, p. 32):

(1)	$(A \Rightarrow ((A \Rightarrow A) \Rightarrow A)) \Rightarrow ((A \Rightarrow (A \Rightarrow A)) \Rightarrow (A \Rightarrow A))$	Instance of A2
(2)	$A \Rightarrow ((A \Rightarrow A) \Rightarrow A)$	Instance of A1
(3)	$(A \Rightarrow (A \Rightarrow A)) \Rightarrow (A \Rightarrow A)$	From 1, 2 by MP
(4)	$A \Rightarrow (A \Rightarrow A)$	Instance of A1
(5)	$A \Rightarrow A$	From 3, 4 by MP

As a second example, let's show that

$$\begin{array}{l} \textit{John is the passenger}, \\ \textit{John is the passenger} \Rightarrow \textit{Betty is the driver}, \\ \textit{Betty is the driver} \Rightarrow \neg\textit{Betty is the passenger} \\ \vdash \neg\textit{Betty is the passenger} \end{array}$$

The derivation is

(1)	*John is the passenger*	Assumption
(2)	*John is the passenger* $\Rightarrow$ *Betty is the driver*	Assumption
(3)	*Betty is the driver*	From 1, 2 by MP
(4)	*Betty is the driver* $\Rightarrow \neg$*Betty is the passenger*	Assumption
(5)	$\neg$*Betty is the passenger*	From 3, 4 by MP

Notice that "rules" in the sense of "rule-based systems" correspond not to rules of inference, but to non-atomic assumptions. Rules of inference are used by the underlying "inference engine" of rule-based systems. We sometimes call assumptions *non-logical axioms* because they are used in proofs the same way axioms are, but they are not actually axioms of any logical system. The rules of rule-based systems are actually non-logical axioms. Sometimes, they are also called *domain rules* to distinguish them from rules of inference.

6.5.2 Natural Deduction Methods

Natural deduction proof methods are characterized by having no axioms, but a Large set of rules of inference[5], including a few **structural rules** of inference, and one **introduction rule** and one **elimination rule** for each connective.

The natural deduction proof method we will present here is based on that of Fitch (1952). A **Fitch-style proof** of a theorem P is

- An ordered list of wfps and subproofs ending with P, such that
- each wfp or subproof on the list must be justified by a rule of inference.

A **Fitch-style derivation** of a wfp P from an assumption A is a hypothetical subproof whose hypothesis is A and which contains

- An ordered list of wfps and inner subproofs ending with P, such that
- each wfp or inner subproof on the list must be justified by a rule of inference.

Structural Rules of Inference

$$\begin{array}{l|ll} i. & A & Hyp \\ \hline & & \end{array}$$

$$\begin{array}{rl|ll} i. & A & & \\ & \vdots & & \\ j. & A & Rep, i & \end{array}$$

$$\begin{array}{rl|ll} i. & A & & \\ & \vdots & & \\ & \vdots & & \\ j. & A & Reit, i & \end{array}$$

Rules for $\neg$

$$\begin{array}{rl|ll} i. & A & Hyp & \\ \hline & \vdots & & \\ j. & B & & \\ j+1. & \neg B & & \\ \hline j+2. & \neg A & & \neg I, i\text{–}(j+1) \end{array}$$

[5] According to (Pelletier, 2000) the real distinguishing feature of natural deduction methods is that they include a method of using subproofs.

$$\begin{array}{ll|ll} i. & \neg\neg A & & \\ j. & A & \neg E, i & \end{array}$$

Rules for $\wedge$

$$\begin{array}{ll|ll} i. & A & & \\ & \vdots & & \\ j. & B & & \\ k. & A \wedge B & \wedge I, i, j & \end{array}$$

$$\begin{array}{ll|ll} i. & A \wedge B & & \\ j. & A & \wedge E, i & \end{array} \qquad \begin{array}{ll|ll} i. & A \wedge B & & \\ j. & B & \wedge E, i & \end{array}$$

Rules for $\vee$

$$\begin{array}{ll|ll} i. & A & & \\ j. & A \vee B & \vee I, i & \end{array} \qquad \begin{array}{ll|ll} i. & B & & \\ j. & A \vee B & \vee I, i & \end{array}$$

$$\begin{array}{rl|ll} i. & A_1 \vee \cdots \vee A_n & & \\ & \vdots & & \\ i + j_1. & A_1 \Rightarrow B & & \\ & \vdots & & \\ i + j_n. & A_n \Rightarrow B & & \\ i + k. & B & \vee E, i, i + j_1, \ldots, i + j_n & \end{array}$$

Rules for $\Rightarrow$

$$\begin{array}{ll|ll} i. & \begin{array}{|l} \hline A \quad Hyp \\ \vdots \\ B \end{array} & & \\ j. & & & \\ k & A \Rightarrow B & \Rightarrow I, i\text{–}j & \end{array} \qquad \begin{array}{ll|ll} i. & A & & \\ & \vdots & & \\ j. & A \Rightarrow B & & \\ k. & B & \Rightarrow E, i, j & \end{array}$$

Rules for $\Leftrightarrow$

$$\begin{array}{ll|ll} i. & A \Rightarrow B & & \\ & \vdots & & \\ j. & B \Rightarrow A & & \\ k. & A \Leftrightarrow B & \Leftrightarrow I, i, j & \end{array}$$

$$\begin{array}{ll|ll} i. & A & & \\ & \vdots & & \\ j. & A \Leftrightarrow B & & \\ k. & B & \Leftrightarrow E, i, j & \end{array} \qquad \begin{array}{ll|ll} i. & B & & \\ & \vdots & & \\ j. & A \Leftrightarrow B & & \\ k. & A & \Leftrightarrow E, i, j & \end{array}$$

Example Fitch-Style Proof

$$\begin{array}{ll|ll} \hline 1. & A & Hyp & \\ \hline 2. & A & Rep, 1 & \\ 3. & A \Rightarrow A & & \Rightarrow I, 1\text{–}2 \end{array}$$

Example Fitch-Style Derivation

1.	*Tom is the driver* $\wedge$(*Tom is the driver* $\Rightarrow$ *Betty is the passenger*) $\wedge$(*Betty is the passenger* $\Rightarrow$ $\neg$*Betty is the driver*)	Hyp
2.	*Tom is the driver*	$\wedge E, 1$
3.	*Tom is the driver* $\Rightarrow$ *Betty is the passenger*	$\wedge E, 1$
4.	*Betty is the passenger*	$\Rightarrow E, 2, 3$
5.	*Betty is the passenger* $\Rightarrow$ $\neg$*Betty is the driver*	$\wedge E, 1$
6.	$\neg$*Betty is the driver*	$\Rightarrow E, 4, 5$

Fitch-Style Proof of Axiom A1

1.	A	Hyp
2.	B	Hyp
3.	A	$Reit, 1$
4.	$B \Rightarrow A$	$\Rightarrow I$, 2–3
5.	$A \Rightarrow (B \Rightarrow A)$	$\Rightarrow I$, 1–4

CarPool World Derivation

1.	$(Tom\ is\ the\ driver \Leftrightarrow \neg Tom\ is\ the\ passenger)$ $\wedge\ (Tom\ is\ the\ passenger \Leftrightarrow Betty\ is\ the\ driver)$ $\wedge\ (Betty\ is\ the\ driver \Leftrightarrow \neg Betty\ is\ the\ passenger)$ $\wedge\ Tom\ is\ the\ driver$	Hyp
2.	$Tom\ is\ the\ driver$	$\wedge E, 1$
3.	$Tom\ is\ the\ driver \Leftrightarrow \neg Tom\ is\ the\ passenger$	$\wedge E, 1$
4.	$\neg Tom\ is\ the\ passenger$	$\Leftrightarrow E, 3$
5.	$\neg Betty\ is\ the\ passenger$	Hyp
6.	$Betty\ is\ the\ driver \Leftrightarrow \neg Betty\ is\ the\ passenger$	$\wedge E, 1$
7.	$Betty\ is\ the\ driver$	$\Leftrightarrow E, 5, 6$
8.	$Tom\ is\ the\ passenger \Leftrightarrow Betty\ is\ the\ driver$	$\wedge E, 1$
9.	$Tom\ is\ the\ passenger$	$\Leftrightarrow E, 7, 8$
10.	$\neg Tom\ is\ the\ passenger$	$Reit, 4$
11.	$\neg\neg Betty\ is\ the\ passenger$	$\neg I$, 5–10
12.	$Betty\ is\ the\ passenger$	$\neg E, 11$

More Connections

- $A \vdash P$ if and only if $\vdash A \Rightarrow P$.
- So proving theorems and deriving conclusions from assumptions are equivalent.
- But no atomic proposition is a theorem.
- So theorem proving makes more use of Introduction Rules than most AI reasoning systems.

3 Important Properties
of Logical Systems

Soundness: $\vdash P$ implies $\models P$

Consistency: not both $\vdash P$ and $\vdash \neg P$

Completeness: $\models P$ implies $\vdash P$

Connections

- If at most 1 of $\models P$ and $\models \neg P$
  then soundness implies consistency.
- Soundness is the essence of "correct reasoning."
- Completeness less important for running systems since a proof may take too long to wait for.
- The Propositional Logics we have been looking at are complete.
- Gödel's Incompleteness Theorem: A logic strong enough to formalize arithmetic is *either* inconsistent *or* incomplete.

$$A_1, \ldots, A_n \vdash P \qquad \Leftrightarrow \qquad \vdash A_1 \wedge \ldots \wedge A_n \Rightarrow P$$

More Connections soundness $\Downarrow\Uparrow$ completeness soundness $\Downarrow\Uparrow$ completeness

$$A_1, \ldots, A_n \models P \qquad \Leftrightarrow \qquad \models A_1 \wedge \ldots \wedge A_n \Rightarrow P$$

Exercises

1. Prove that $\models (A \Rightarrow B) \vee (B \Rightarrow A)$.
2. Prove that $\models \neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$.
3. Letting Γ be the domain rules for CarPool World, show that
   $\Gamma, \mathit{Betty\ drives\ Tom} \models \neg \mathit{Betty\ is\ the\ passenger} \wedge \neg \mathit{Tom\ is\ the\ driver}$.
4. This question was adapted from S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach, Second Edition,* (Upper Saddle River, NJ: Pearson Education, Inc.), 2003, p. 238.

   (a) (3) Formalize the domain that has the following domain rules, by showing the syntax and intensional semantics of the atomic propositions.

   > "If the unicorn is mythical, then it is immortal, but if it is not mythical, then it is a mortal mammal. If the unicorn is either immortal or a mammal, then it is horned. The unicorn is magical if it is horned."

   (b) Using your formalization, list the wfps that represent the above domain rules.

   (c) Using the `wang` program, determine which of the following sentences is logically implied by these domain rules.

   i. The unicorn is mythical.
   ii. The unicorn is magical.
   iii. The unicorn is horned.

Chapter 7

Clause Form Logic

7.1 Syntax

7.2 Semantics

7.3 Proof Theory

Part III

Predicate Logic Over Finite Models

Chapter 8

The "Standard" Logic

8.1 Syntax

8.2 Semantics

An interesting discussion of substitutional *vs.* the usual, objectual, semantics is in (Hand, 2007).

8.3 Proof Theory

Chapter 9

Clause Form Logic

9.1 Syntax

9.2 Semantics

9.3 Proof Theory

Chapter 10

SNePS Logic

10.1 Syntax

10.2 Semantics

10.3 Proof Theory

Part IV

Full First-Order Predicate Logic

Chapter 11

The "Standard" Logic

11.1 Syntax

11.2 Semantics

11.3 Proof Theory

Chapter 12

Clause Form Logic

12.1 Syntax

12.2 Semantics

12.3 Proof Theory

Chapter 13

SNePS Logic

13.1 Syntax

13.2 Semantics

13.3 Proof Theory

Part V

Relevance Logic

Bibliography

Beth, E. W. (1959). *The Foundations of Mathematics*. North Holland, Amsterdam.

Bransford, J. D. and Franks, J. J. (1971). The abstraction of linguistic ideas. *Cognitive Psychology*, 2(4):331–350.

Fitch, F. B. (1952). *Symbolic Logic: An Introduction*. Ronald Press, New York.

Haack, S. (1978). *Philosophy of Logics*. Cambridge University Press, New York.

Hand, M. (2007). Objectual and substitutional interpretations of the quantifiers. In Jacquette, D., editor, *Philosophy of Logic*, Handbook of the Philosophy of Science. Elsevier B.V., Amsterdam.

ISO/IEC (2007). *Information technology — Common Logic (CL): a framework for a family of logic-based languages, ISO/IEC 24707:2007(E)*. ISO/IEC, Switzerland, First edition. available from `http://standards.iso/ittf/license.html`.

Lifschitz, V., editor (1990). *Formalizing Common Sense: Papers by John McCarthy*. Ablex Publishing Corporation, Norwood, NJ.

McCarthy, J. (1959). Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, London. Her Majesty's Stationery Office. Reprinted in (Lifschitz, 1990, 9–16).

McCawley, J. D. (1981). *Everything that Linguists have Always Wanted to Know about Logic* *but were ashamed to ask*. The University of Chicago Press, Chicago, IL.

Mendelson, E. (1964). *Introduction to Mathematical Logic*. D. Van Nostrand, Princeton, NJ.

Mueller, E. T. (2006). *Commonsense Reasoning*. Elsevier, Amsterdam.

Mueller, E. T. (2008). *Discrete Event Calculus Reasoner Documentation*. IBM Thomas J. Watson Research Center, Yorktown Heights, NY. available at `http://decreasoner.sourceforge.net/csr/decreasoner.pdf` accessed December 2, 2008.

Pelletier, F. J. (2000). A history of natural deduction and elementary logic textbooks. In Woods, J. and Brown, B., editors, *Logical Consequence: Rival Approaches*, volume 1, pages 105–138. Hermes Science Pubs, Oxford.

Rapaport, W. J. (1992). Logic. In (Shapiro, 1992b), pages 851–853.

Shapiro, S. C. (1987). Processing, bottom-up and top-down. In Shapiro, S. C., editor, *Encyclopedia of Artificial Intelligence*, pages 779–785. John Wiley & Sons, New York. Reprinted in Second Edition, 1992, pages 1229–1234.

Shapiro, S. C. (1992a). Artificial intelligence. In (Shapiro, 1992b), pages 54–57.

Shapiro, S. C., editor (1992b). *Encyclopedia of Artificial Intelligence*. John Wiley & Sons, New York, second edition.

Shapiro, S. C. (2000). Propositional, first-order and higher-order logics: Basic definitions, rules of inference, and examples. In Iwańska, Ł. M. and Shapiro, S. C., editors, *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*, pages 379–395. AAAI Press/The MIT Press, Menlo Park, CA.

Shapiro, S. C. (2003). Knowledge representation. In Nadel, L., editor, *Encyclopedia of Cognitive Science*, volume 2, pages 671–680. Macmillan Publishers Ltd.

Siekmann, J. and Wrightson, G., editors (1983). *Classical Papers on Computational Logic 1957–1966*, volume 1 of *Automation of Reasoning*. Springer-Verlag, Berlin.

Smullyan, R. M. (1968). *First-Order Logic*. Spring-Verlag, New York.

Wang, H. (1960). Toward mechanical mathematics. *IBM Journal of Research and Development*, 4:2–22. Reprinted in (Siekmann and Wrightson, 1983, 244–264).