# Preface

## Goals

Keeping up with new developments in most areas of computing requires familiarity with basic logical concepts. In particular, your success in most aspects of software development significantly depends on your ability to reason correctly, to communicate your reasoning, and to understand and evaluate the reasoning of others. These abilities are critical for anyone who does feasibility analysis, systems analysis, problem specification, database design or management, program design, coding, testing, verification, problem diagnosis, documentation, software maintenance, or research in any of these areas.

If you know little about how logic can be used in software development and if you want to know more, then this book may be of use to you. After reading it you should be better able to reason about software development, to communicate your reasoning, to distinguish between good and bad reasoning, and to read professional literature, which presumes knowledge of elementary logic.

On the other hand, if you think that your own logical abilities are good enough, but that many other people are sadly deficient in these abilities, then please give copies of this book to those who need it.

## Overview and Features

Applications of logic to software development are emphasized throughout. Examples involving program instructions are expressed in pseudocode so that the book makes no use of any particular programming language. It is divided into three parts. Part I is about language and logical form. It explains how to find and represent the logical forms of statements expressed in English. It shows how to use a subset of English, here called *logical English*, to represent both the meanings and logical forms of statements. Logical English is intermediate between informal but meaningful English and the largely meaningless and severely abstract notations commonly used in formal logic, and used here in Part III. This intermediate role resembles the role of pseudocode. Like pseudocode, logical English is still recognizable English. And like pseudocode, it is a helpful bridge between informal English and a highly formal notation. They differ in that logical English is used to express statements and conditions while pseudocode is used to express instructions. Part I ends by describing how to use logical English to clarify and express data structure definitions, problem specifications, and conditions in instructions.

Part II is about truth in the ordinary "material" sense of that term. It shows how to use truth tables to determine the truth or falsity of a complex statement built using connectives such as "not", "and", "or", and "if…then…" if you know the truth or falsity of its component statements. Truth conditions for statements involving the quantifiers "all" and "some" are also described. Following that, several computer-related applications of this material are discussed. The final chapter shows how to apply truth value calculations to forward and backward tracing of program execution.

Part III is about "logical" truth. Logical truth is a generalization of material truth. It involves ignoring the meanings and material truth values of individual statements and focusing only on their logical forms. Much of what is known about how to reason correctly can best be stated in terms of logical forms. For example, the statement form "P or not P" is logically true. As a result, no matter what statement is used in place of P, the resulting statement of the form "P or not P" is materially true.

This part also explains and shows how to test statements for logical equivalence, logical implication, and logical redundancy, and how to test arguments for validity and soundness. It also explains how to use rules of inference to make proofs. It then describes how to apply these concepts to problem specifications. This is followed by a proof that no computer program that solves the problem of determining whether any arbitrarily selected program will halt with any arbitrarily selected input can be written. That bad news is followed by the good news that it is possible, though difficult, to prove that a program is correct relative to a problem specification, without doing any testing. Examples showing how to do this in simple cases are given. The last chapter briefly discusses some topics not covered here, e.g. logic testing and quantum computing. It includes a few pointers to additional sources of information about these topics.

## Suggested Uses

This book is designed to be used by computer professionals and students who want to study on their own without an instructor. It is also suitable as the primary text for instructor led introductory courses on logic for students who are studying any of the computing disciplines. Earlier versions of much of this material were class tested in a college level course I teach on logic and its applications to computing. In addition, the three parts of the book can be the basis for three or more short professional development courses.

## Target Audiences

Students and professionals who expect to be involved with any aspect of software development are the target audiences for this book. No prior knowledge of formal logic is assumed. Some knowledge of software development is assumed.

## Audience Resources

Many examples as well as many practice exercises, along with solutions to half of them, are included here. Solutions to all the exercises can be had by instructors at http://www.springer.com/978-1-84800-081-0

Readers can contact me at http://www.logicforsoftwaredevelopment.com. I intend to post corrections to newly discovered errors, pointers to additional resources, logic jokes, and other related information there.

Readers are also urged to use the email link there to send error reports and constructive suggestions for improving this book.

## Acknowledgements

I am grateful to the monks, administrators, staff, and faculty at Belmont Abbey College for providing me an environment in which it was possible to write this book. Special thanks, and some commiseration, are due to my students in CS325 who suffered through earlier drafts of much of this material.

Gene Lauver and Bill Seltzer, who have many years of experience in real-world computing, found numerous errors, made many helpful suggestions, and even worked many of the exercises "for fun". Without their help, this book would have been noticeably less useful.

Finally, several anonymous reviewers made helpful suggestions. Wayne Wheeler and Catherine Brett at Springer, UK, provided enthusiastic, cheerful, patient, and professional editorial support.

Robert Lover
Charlotte, NC

# Chapter 2
# Compound Statements

This chapter describes logical English for statements that are made from one or more simpler statements. Such statements are called compound statements. They are formed using words and phrases called statement connectives. Some of those connectives are said to be *truth functional*. Logical English abbreviations for the most important truth functional statement connectives are introduced here. The role of parentheses to reduce ambiguity is also discussed. After studying this material you should be able to transform truth functional compound statements between (ordinary) English and logical English and use parentheses and conventions for dropping them.

## Outline

2.1 Truth functional connectives
2.2 Statements with multiple connectives
2.3 Parenthesis dropping conventions

## 2.1 Truth Functional Connectives

*Definition 1.* A *statement connective* is a word or phrase used to construct complex statements out of simpler statements.

*Example 1.* The statement connective “not” and the statement “John is tall.” can be used to construct the more complex statement “John is not tall.” The connective “and” can be used with the statements “John is tall” and “Carol is thin.” to form the statement “John is tall and Carol is thin.”

There are many statement connectives in English. The ones that are important here are said to be ‘truth functional’. A statement connective is truth functional if and only if the truth value (true or false) of a compound statement made using it can be determined from knowledge of just the truth values of its component statements, without knowing anything about their meanings. Truth functionality will be discussed in detail in later chapters. In logical English symbols are often used in place of truth functional connectives. Several different sets of symbols for connectives are in common use. The set that will be used here is described below.

| Approximate English Meaning | Logic Symbol | Name of Symbol |
|---|---|---|
| not | $\sim$ | tilde |
| and | $\wedge$ | up wedge |
| inclusive or (i.e. and/or) | $\vee$ | wedge |
| if…then… | $\rightarrow$ | right arrow |
| if and only if | $\leftrightarrow$ | double arrow |

In the following definition L represents the statement on the left, and R represents the statement on the right. Note that L and R can themselves be atomic or compound.

*Definition 2.* Suppose L and R represent statements. Then:

(a) ‘Not R’ is called *the negation of R*, abbreviated ‘(~R)’.
(b) ‘L and R’ is called *the conjunction of L with R*, abbreviated ‘(L $\wedge$ R)’. L and R are its *conjuncts*.
(c) ‘L or R’ is called *the disjunction of L with R*, abbreviated ‘(L $\vee$ R)’. L and R are its *disjuncts*.
(d) ‘If L then R’ is called *the conditional of L with R*, abbreviated ‘(L $\rightarrow$ R)’. L is called the *antecedent* of the conditional, and R is called the *consequent* of the conditional.
(e) ‘L if and only if R’ is called *the biconditional (or the equivalence) of L with R*, abbreviated ‘(L $\leftrightarrow$ R)’. L and R are its *components*.

Recall that in Chapter 1 atomic statements could be represented in logical English by single capital letters followed by a list of names and definite descriptions in parentheses. In this chapter we are not concerned with the names and descriptions. Consequently, here both atomic and compound statements will often be represented by single capital letters of the alphabet, without lists in parentheses. The use of single capital letters without lists in parentheses is not necessary. It is simply a space saving measure. Moreover, it is not necessary to use the abbreviations given above for sentential connectives. In real life, things are usually more complicated than in the examples given here and space saving may be less important than remembering what abbreviation goes with which English statement. In those cases you may be better off using less abbreviated notations.

*Example 2.* Logical English for connectives

| Grammatical category | English | Logical English |
|---|---|---|
| atomic statement | John is asleep. | A<br>S(j)<br>JSleep |
| atomic statement | Today is Monday. | M<br>Mon |
| atomic statement | d = 7. | D<br>D7 |
| negation | John is not asleep. | (~A)<br>(not A)<br>(~S(j))<br>(~JSleep) |
| conjunction | John is asleep and today is Monday. | (A $\wedge$ M)<br>(S(j) $\wedge$ Mon)<br>(A and M)<br>(JSleep $\wedge$ Mon) |
| conditional | If today is Monday then d = 7. | (M $\rightarrow$ D)<br>(Mon $\rightarrow$ D7) |
| conditional | If John is asleep then today is Monday. | (A $\rightarrow$ M)<br>(If A then M)<br>(JSleep $\rightarrow$ Mon) |
| equivalence | John is asleep if and only if today is Monday. | (A $\leftrightarrow$ M)<br>(Jsleep $\leftrightarrow$ Mon) |

Learning to represent the logical structure of compound English statements using logical English is best done by seeing many examples and then practicing yourself.

*Example 3.* Logical English for different English statements

Recall that in English there are usually many different ways to say approximately the same thing, i.e. there are many sentences that have approximately the same meaning. Consequently, even if two sentences have slightly different meanings, they may be represented by the same logical English abbreviation, provided that the difference in meaning does not affect truth values. For example, ‘It is raining but I am dry.’ can be represented by the same abbreviation as ‘It is raining and I am dry.’ because the slight difference in meaning between the two sentences has no affect on how the truth value of either depends on the truth value of ‘It is raining’ and the truth value of ‘I am dry’.

Suppose that M represents ‘Today is Monday.’ and T represents ‘Taxes are due.’ Some sentences that can be represented by ‘(~M)’ are:

(a) Today is not Monday.
(b) It is not the case that today is Monday.
(c) It is false that today is Monday.

Some English statements that can be represented by ‘(M ∧ T)’ are:

(a) Today is Monday and taxes are due.
(b) It is the case that today is Monday and that taxes are due.
(c) Today is Monday even though taxes are due.
(d) Although today is Monday, taxes are due.
(e) Today is Monday but taxes are due.
(f) Today is Monday although taxes are due.

Note that ‘Taxes are due and today is Monday’ would be represented by (T ∧ M), not by (M ∧ T). The order in which things are said is often important and should be preserved where possible, even when the meaning of two differently ordered abbreviations is the same.

Some English sentences that can be represented by ‘(M ∨ T)’ are:

(a) Today is Monday or taxes are due.
(b) It is the case that today is Monday or that taxes are due.
(c) Either today is Monday or taxes are due or both.

In English there are two logically different uses of ‘or’, the inclusive use and the exclusive use. Some languages have two separate words for these two meanings. In logical English the word ‘or’ and the symbol ‘∨’ are used to represents the inclusive use where ‘P ∨ Q’ means ‘P or Q or both P and Q’. The English word ‘xor’ is sometimes used to represent the exclusive sense of ‘or’ as in ‘P or Q but not both P and Q’. It can also be expressed by ‘(P ∨ Q) ∧ ~(P ∧ Q)’.

Some English sentences that can be represented by ‘(M → T)’ are:

(a) If today is Monday then taxes are due.
(b) If today is Monday, taxes are due.
(c) Provided that today is Monday, taxes are due.
(d) Taxes are due if today is Monday.
(e) In case today is Monday, taxes are due.

In general, if ‘(M → T)’ is true then we say that M is a sufficient condition for T and that T is a necessary condition for M. Some more English sentences that can be represented by ‘(M → T)’ are:

(f) Today being Monday is a sufficient condition for taxes to be due.
(g) Taxes being due is a necessary condition for today to be Monday.

Some English sentences that can be represented by ‘(M ↔ T)’ are:

(a) Today is Monday if and only if taxes are due
(b) Today’s being Monday is a necessary and sufficient condition for taxes being due.
(c) Today is Monday just in case taxes are due.

## 2.2 Statements with Multiple Connectives

Ambiguity can result when a statement has more than one connective if the scope to which each connective applies is not clear. For example, a statement of the form 'P and Q or R' could be understood as 'P and (Q or R)' or as '(P and Q) or R'. These two forms are not equivalent. Parentheses can be used to disambiguate otherwise ambiguous statement forms. In logic parentheses are used in the same way they are used in mathematics, i.e. work from the inside out.

*Example 4.* Statements with multiple connectives

| English | Logical English |
|---|---|
| Today is Monday. | M |
| Taxes are due. | T |
| Today is Friday. | F |
| Joe is happy. | H |
| Joe is broke. | B |
| (a) Today is Monday or Today is Friday, but not both. | $((M \vee F) \wedge {\sim}(M \wedge F))$ |
| (b) If today is Monday then today is not Friday. | $M \rightarrow ({\sim}F))$ |
| (c) If today is not Monday then Joe is not happy. | $(({\sim}M) \rightarrow ({\sim}H))$ |
| (d) Joe is happy if and only if taxes are not due. | $(H \leftrightarrow ({\sim}T))$ |
| (e) If Joe is happy and Joe is broke then taxes are not due. | $((H \wedge B) \rightarrow ({\sim}T))$ |
| (f) If Joe is broke then Joe is happy just in case today is Friday. | $(B \rightarrow (H \leftrightarrow F))$ |
| (g) If today is neither Monday nor Friday then Joe is neither happy nor broke. | $((({\sim}M) \wedge ({\sim}F)) \rightarrow (({\sim}H) \wedge ({\sim}B)))$ |

*Exercise 1.* For each part, identify the missing grammatical category and use the logic notation described below to transform the English statements into logical English.

| Grammatical category | English | Logical English |
|---|---|---|
| atomic statement | a = 3. | A |
| atomic statement | b = 5. | B |
| atomic statement | a + b = 8. | C |
| (a) | not (a = 3). | |
| (b) | not (b = 5). | |
| (c) | a = 3 and b = 5. | |
| (d) | a = 3 or b = 5. | |
| (e) | If a = 3 then b = 5. | |
| (f) | a = 3 if and only if b = 5. | |
| (g) | If a = 3 or b = 5 then a + b = 8. | |
| (h) | not not b = 5. | |
| (i) | If a not = 3 and b not = 5 then a + b not = 8. | |

A single statement expressed in English can be represented in logical English in different ways, depending on how much detail is represented. Each of the logical English representations shown below is correct. Which one to use depends upon how much detail is relevant at the time.

*Example 5.* Alternative representations

| English | Logical English representations | |
|---|---|---|
| Jack will leave early if and only if the boss is not here. | L | (minimal detail) |
| | (J ↔ B) | (more detail) |
| | (J ↔ (~E)) | (yet more detail) |
| | (LeaveEarly(j) ↔ (~Here(b)) | |

If transformation from English to logical English is so indeterminate, you might wonder why anyone would do it. The short answer is that it is for the same reason we do accounting with Arabic numerals and mathematical symbols rather than writing it all out longhand in English. Imagine writing "A deposit of thirty seven dollars and eighty two cents added to a previous balance of two hundred forty dollars and seventeen cents gives a new balance of ...." in order to balance your checkbook. The mathematical notation, even though it is not unique and it takes time to learn to use, makes doing mathematics much much easier. Using logical English notation has similar advantages if you want to clearly express and reason correctly about almost anything, including computing related issues.

## 2.3 Parenthesis Dropping Conventions

Because complex English statements can lead to logical English expressions having confusingly many pairs of parentheses, it is often helpful to use parenthesis dropping conventions similar to those used in mathematics. Recall for example that in mathematics exponentiation has higher precedence than multiplication and division and they have higher precedence than addition and subtraction. In other words, exponentiation is done before multiplication and division which are done before addition and subtraction, so that 3 + 5 * 7 means 3 + (5 * 7) and not (3 + 5) * 7. Similarly, $3^2 + 7 * 5^2$ means $(3^2) + (7 * (5^2))$.

*LE Rule 3.* In addition to the parenthesis dropping conventions of mathematics, the following *parenthesis dropping conventions* (also called *precedence rules*) will be used for logical English.

(a) ~ has the highest precedence of all.
(b) $\wedge$ , $\vee$ , $\rightarrow$, and $\leftrightarrow$ have successively lower precedence.
(c) Matching pairs of parentheses can be removed if doing so does not cause ambiguity as to how to restore them. In particular, the outermost pair of parentheses may be removed.

*Example 6.* Application of LE Rule 3.

| Fully parenthesized | With parentheses dropping conventions | Part of LE3 used |
|---|---|---|
| (a) (~A) | ~A | c |
| (b) (~(~A)) | ~~A | c twice |
| (c) ((~A) ∨ (~B)) | (~A) ∨ (~B) | c |
| | ~A ∨ ~B | a |
| (d) (((~A) ∨ (~B)) → (~C)) | ((~A) ∨ (~B)) → (~C) | c |
| | (~A) ∨ (~B) → (~C) | b |
| | ~A ∨ ~B → ~C | a |
| (e) (~A) ∨ ((~B) → (~C)) | ~A ∨ (~B → ~C) | a |
| | but not ~A ∨ ~B → ~C | not allowed! |
| (f) ((A ∧ B) ∨ C) | (A ∧ B) ∨ C | c |
| | A ∧ B ∨ C | b |
| (g) (A ∧ (B ∨ C)) | A ∧ (B ∨ C) | c |
| (h) (A ∨ (B ∨ (C ∨ D))) | A ∨ (B ∨ (C ∨ D)) | c |
| | but not A ∨ B ∨ C ∨ D | not allowed |
| (i) (((A ∨ B) ∨ C) ∨ D) | ((A ∨ B) ∨ C) ∨ D | c |
| (j) (~(A ∨ (B ↔ C))) | ~(A ∨ (B ↔ C)) | c |
| (k) ((((~A) ∧ B) → C) ↔ D) | (((~A) ∧ B) → C) ↔ D | c |
| | ((~A ∧ B) → C) ↔ D | a |
| | (~A ∧ B → C) ↔ D | b |
| | ~A ∧ B → C↔D | b |
| (l) (~(A ∧ (B → (C ↔ D)))) | ~(A ∧ (B → (C ↔ D))) | c |

*Exercise 2.* Fully restore parentheses to the following logical English notations. Suggestion: work from highest to lowest precedence, in steps.

(a) P ∨ Q ∧ R
(b) P ∧ Q ∨ R
(c) P → Q ∨ R
(d) P ∨ Q → R
(e) (P ∨ Q) ∨ R
(f) P ∨ (Q ∨ R)
(g) ~P → ~Q ∨ R
(h) ~(P → Q) ∨ R
(i) ~P ∧ Q ∨ R → S ↔ T
(j) P ↔ Q → R ∨ S ∧ ~T

*Exercise 3.* Use the following statement letters to transform each of the English statements below into logical English. Use parenthesis dropping.

| English | Logical English |
|---|---|
| The program compiled correctly. | P |
| The file was sorted. | S |
| The file was corrupted. | C |
| There was an error in the sort routine. | E |

| English | Logical English |
|---|---|
| The program ran correctly. | R |
| The error flag was set at line 4008. | F |
| a < b | L |

(a) If the program ran correctly then the file was sorted.
(b) If there was an error in the sort routine then the file was not sorted.
(c) If the error flag was set at line 4008 then the file was corrupted.
(d) The program compiled correctly and the file was sorted just in case the program ran correctly and there was no error in the sort routine.
(e) A sufficient condition for the file being corrupted is that the error flag was set at line 4008.
(f) A necessary condition for the file being corrupted is that the error flag was set at line 4008.
(g) A necessary and sufficient condition for the file being corrupted is that the error flag was set at line 4008.
(h) The file was sorted unless the program did not run correctly.
(i) If the program compiled correctly and the file was sorted then the program ran correctly or a < b.
(j) a < b if and only if the program did not run correctly or the error flag was not set at line 4008.
(k) If a < b and the file was sorted correctly then the program ran correctly if and only if there was not an error in the sort routine.

*Exercise 4.* Use the following statement letters to transform each of the Logical English statements below into English.

| Logical English | English |
|---|---|
| P | The program compiled correctly. |
| S | The file was sorted. |
| C | The file was corrupted. |
| E | There was an error in the sort routine. |
| R | The program ran correctly. |
| F | The error flag was set at line 4008. |
| L | a < b. |

(a) $R \rightarrow P$
(b) $\sim\sim R$
(c) $\sim P \rightarrow \sim R$
(d) $F \rightarrow L$
(e) $C \vee E \rightarrow \sim P$
(f) $P \wedge \sim S \rightarrow C$

*Exercise 5.* Use the following statement letters to transform each of the English statements below into logical English.

| English | Logical English |
|---|---|
| 6 is a domain value of the problem. | S |
| 1 is a domain value of the problem. | O |
| 0 is the solution value of the problem. | N |
| The domain data is sorted small to large. | A |
| The domain data is sorted large to small. | D |

(a) If 1 is a domain value of the problem then the domain data is not sorted small to large or large to small.
(b) If the domain data is sorted small to large then the domain data is not sorted large to small.
(c) If a domain value of the problem is not 6 and is not 1 then the domain data is not sorted.
(d) If 6 is a domain value of the problem then 1 is not a domain value of the problem.
(e) If the domain data is sorted large to small or small to large then the solution value of the problem is 0.
(f) If the domain data is not sorted large to small and not sorted small to large then the solution value of the problem is 0.
(g) A sufficient condition for the solution value of the problem to be 0 is that a domain value of the problem is 6 if and only if the domain data is sorted small to large.
(h) If a domain value of the problem is 6 then 0 is not the solution value of the problem unless the domain data is sorted small to large.

*Exercise 6.* Use the following statement letters to transform each of the logical English statements below into English.

| Logical English | English |
|---|---|
| S | 6 is a domain value of the problem. |
| O | 1 is a domain value of the problem. |
| N | 0 is the solution value of the problem. |
| A | The domain data is sorted small to large. |
| D | The domain data is sorted large to small. |

(a) $N \vee {\sim}N$
(b) $A \leftrightarrow {\sim}D$
(c) $S \rightarrow {\sim}O$
(d) ${\sim}S \wedge {\sim}O \rightarrow {\sim}N$
(e) $N \rightarrow S \vee O$
(f) ${\sim}(S \wedge O) \rightarrow {\sim}S \vee {\sim}O$