Part 0
Why Use Logic?
Why Prove Programs Correct?

A story

We have just finished writing a large program (3000 lines). Among other things, the program computes as intermediate results the quotient q and remainder r arising from dividing a non-negative integer x by a positive integer y. For example, with $x = 7$ and $y = 2$, the program calculates $q = 3$ (since $7 \div 2 = 3$) and $r = 1$ (since the remainder when 7 is divided by 2 is 1).

Our program appears below, with dots "..." representing the parts of the program that precede and follow the remainder-quotient calculation. The calculation is performed as given because the program will sometimes be executed on a micro-computer that has no integer division, and portability must be maintained at all costs! The remainder-quotient calculation actually seems quite simple; since $\div$ cannot be used, we have elected to repeatedly subtract divisor y from a copy of x, keeping track of how many subtractions are made, until another subtraction would yield a negative integer.

```
. . .
r:= x; q:= 0;
while r > y do
  begin r:= r-y; q:= q+1 end;
. . .
```

We're ready to debug the program. With respect to the remainder-quotient calculation, we're smart enough to realize that the divisor should initially be greater than 0 and that upon its termination the variables should satisfy the formula

$x = y*q + r$,

so we add some output statements to check the calculations:

```
. . .
write('dividend x =', x, 'divisor y =', y);
r := x; q := 0;
while r > y do
  begin r := r-y; q := q+1 end;
write('y*q + r =', y*q + r);
. . .
```

Unfortunately, we get voluminous output because the program segment occurs in a loop, so our first test run is wasted. We try to be more selective about what we print. Actually, we need to know values only when an error is detected. Having heard of a new feature just inserted into the compiler, we decide to try it. If a Boolean expression appears within braces { and } at a point in the program, then, whenever "flow of control" reaches that point during execution, it is checked: if false, a message and a dump of the program variables are printed; if true, execution continues normally. These Boolean expressions are called *assertions*, since in effect we are asserting that they should be true when flow of control reaches them. The systems people encourage leaving assertions in the program, because they help document it.

Protests about inefficiency during production runs are swept aside by the statement that there is a switch in the compiler to turn off assertion checking. Also, after some thought, we decide it may be better to always check assertions —detection of an error during production would be well worth the extra cost.

So we add assertions to the program:

```
    . . .
    {y > 0}
    r := x; q := 0;
(1) while r > y do
      begin r := r-y; q := q+1 end;
    {x = y*q + r}
    . . .
```

Testing now results in far less output, and we make progress. Assertion checking detects an error during a test run because y is 0 just before a remainder-quotient calculation, and it takes only four hours to find the error in the calculation of y and fix it.

But then we spend a day tracking down an error for which we received no nice false-assertion message. We finally determine that the remainder-quotient calculation resulted in

$x = 6, y = 3, q = 1, r = 3$.

Sure enough, both assertions in (1) are true with these values; the problem is that the remainder should be less than the divisor, and it isn't. We determine that the loop condition should be $r \geqslant y$ instead of $r > y$. If only the result assertion were strong enough —if only we had used the assertion $x = y*q + r$ **and** $r < y$— we would have saved a day of work! Why didn't we think of it?

We fix the error and insert the stronger assertion:

```
. . .
{y > 0}
r := x; q := 0;
while r ≥ y do
  begin r := r-y; q := q+1 end;
{x = y*q + r and r < y}
. . .
```

Things go fine for a while, but one day we get incomprehensible output. It turns out that the quotient-remainder algorithm resulted in a negative remainder $r = -2$. But the remainder shouldn't be negative! And we find out that r was negative because initially x was -2. Ahhh, another error in calculating the input to the quotient-remainder algorithm —x isn't *supposed* to be negative! But we could have caught the error earlier and saved two days searching, in fact we *should* have caught it earlier; all we had to do was make the initial and final assertions for the program segment strong enough. Once more we fix an error and strengthen an assertion:

```
. . .
{0 ≤ x and 0 < y}
r := x; q := 0;
while r ≥ y do
  begin r := r-y; q := q+1 end;
{x = y*q + r and 0 ≤ r < y}
. . .
```

It sure would be nice to be able to invent the right assertions to use in a less *ad hoc* fashion. Why can't we think of them? Does it have to be a trial-and-error process? Part of our problem here was carelessness in specifying what the program segment was to do —we should have written the initial assertion ($0 \leqslant x$ **and** $0 < y$) and the final assertion ($x = y*q + r$ **and** $0 \leqslant r < y$) *before* writing the program segment, for they form the definition of quotient and remainder.

But what about the error we made in the condition of the while loop? Could we have prevented that from the beginning? Is there is a way to prove, just from the program and assertions, that the assertions are true when flow of control reaches them? Let's see what we can do.

Just before the loop it seems that part of our result,

(2) $x = y*q + r$

holds, since $x = r$ and $q = 0$. And from the assignments in the loop body we conclude that if (2) is true before execution of the loop body then it is true after its execution, so it will be true just before and after *every* iteration of the loop. Let's insert it as an assertion in the obvious places, and let's also make all assertions as *strong* as possible:

```
. . .
{0≤x and 0<y}
r:= x; q:= 0;
{0≤r and 0<y and x =y*q +r}
while r ≥y do
   begin {0≤r and 0<y ≤r and x =y*q +r}
         r:= r−y; q:= q+1
         {0≤r and 0<y and x =y*q +r}
end;
{0≤r <y and x = y*q +r}
. . .
```

Now, how can we easily determine a correct loop condition, or, given the condition, how can we prove it is correct? When the loop terminates the condition is false. Upon termination we want $r < y$, so that the complement, $r \geqslant y$ must be the correct loop condition. How easy that was!

It seems that if we knew how to make all assertions as strong as possible and if we learned how to reason carefully about assertions and programs, then we wouldn't make so many mistakes, we would *know* our program was correct, and we wouldn't need to debug programs at all! Hence, the days spent running test cases, looking through output and searching for errors could be spent in other ways.

Discussion

The story suggests that assertions, or simply Boolean expressions, are really needed in programming. But it is not enough to know how to write Boolean expressions; one needs to know how to *reason* with them: to simplify them, to prove that one follows from another, to prove that one is not true in some state, and so forth. And, later on, we will see that it is necessary to use a kind of assertion that is not part of the usual Boolean expression language of Pascal, PL/I or FORTRAN, the "quantified" assertion.

Knowing how to reason about assertions is one thing; knowing how to reason about *programs* is another. In the past 10 years, computer science has come a long way in the study of proving programs correct. We are reaching the point where the subject can be taught to undergraduates, or to anyone with some training in programming and the will to become more proficient. More importantly, the study of program correctness proofs has led to the discovery and elucidation of methods for *developing* programs. Basically, one attempts to develop a program and its proof hand-in-hand, with the proof ideas leading the way! If the methods are practiced with care, they can lead to programs that are free of errors, that take much less time to develop and debug, and that are much more easily understood (by those who have studied the subject).

Above, I mentioned that programs could be free of errors and, in a way, I implied that debugging would be unnecessary. This point needs some clarification. Even though we can become more proficient in programming, we will still make errors, even if only of a syntactic nature (typos). We are only human. Hence, some testing will always be necessary. But it should not be called debugging, for the word debugging implies the existence of bugs, which are terribly difficult to eliminate. No matter how many flies we swat, there will always be more. A disciplined method of programming should give more confidence than that! We should run test cases not to look for bugs, but to increase our confidence in a program we are quite sure is correct; finding an error should be the exception rather than the rule.

With this motivation, let us turn to our first subject, the study of logic.

Part I
Propositions and Predicates

Chapter 1 defines the syntax of propositions —Boolean expressions using only Boolean variables— and shows how to evaluate them. Chapter 2 gives rules for manipulating propositions, which is often done in order to find simpler but equivalent ones. This chapter is important for further work on programming, and should be studied carefully.

Chapter 3 introduces a *natural deduction system* for proving theorems about propositions, which is supposed to mimic in some sense the way we "naturally" argue. Such systems are used in research on mechanical verification of proofs of program correctness, and one should become familiar with them. But the material is not needed to understand the rest of the book and may be skipped entirely.

Chapter 4 extends propositions to include variables of types besides Boolean and introduces quantification. A *predicate calculus* is given, in which one can express and manipulate the assertions we make about program variables. "Bound" and "free" variables are introduced and the notion of textual substitution is studied. This material is necessary for further reading.

Chapter 5 concerns arrays. Thinking of an array as a function from subscript values to array element values, instead of as a collection of independent variables, leads to some neat notation and rules for dealing with arrays. The first two sections of this chapter should be read, but the third may be skipped on first reading.

Finally, chapter 6 discusses briefly the use of assertions in programs, thus motivating the next two parts of the book.

Chapter 1
Propositions

We want to be able to describe sets of states of program variables and to write and manipulate clear, unambiguous assertions about program variables. We begin by considering only variables (and expressions) of type *Boolean*: from the operational point of view, each variable contains one of the values T and F, which represent our notions of "truth" and "falsity", respectively. The word *Boolean* comes from the name of a 19th century English mathematician, George Boole, who initiated the algebraic study of truth values.

Like many logicians, we will use the word *proposition* for the kind of Boolean or logical expression to be defined and discussed in this chapter.

Propositions are similar to arithmetic expressions. There are operands, which represent the values T or F (instead of integers), and operators (e.g. **and**, **or** instead of *, +), and parentheses are used to aid in determining order of evaluation. The problem will not be in defining and evaluating propositions, but in learning how to express assertions written in English as propositions and to reason with those propositions.

1.1 Fully Parenthesized Propositions

Propositions are formed according to the following rules (the operators will be defined subsequently). As can be seen, parentheses are required around each proposition that includes an operation. This restriction, which will be weakened later on, allows us to dispense momentarily with problems of precedence of operators.

1. T and F are propositions.
2. An identifier is a proposition. (An identifier is a sequence of one or more digits and letters, the first of which is a letter.)

3. If b is a proposition, then so is $(\neg b)$.
4. If b and c are propositions, then so are $(b \wedge c)$, $(b \vee c)$, $(b \Rightarrow c)$, and $(b = c)$.

This syntax may be easier to understand in the form of a BNF grammar (Appendix 1 gives a short introduction to BNF):

```
(1.1.1) <proposition> ::= T | F | <identifier>
                        | ( ¬ <proposition> )
                        | ( <proposition> ∧ <proposition> )
                        | ( <proposition> ∨ <proposition> )
                        | ( <proposition> ⇒ <proposition> )
                        | ( <proposition> = <proposition> )
```

Example. The following are propositions (separated by commas):

F, $(\neg T)$, $(b \vee xyz)$, $((\neg b) \wedge (c \Rightarrow d))$,
$((abc1 = id) \wedge (\neg d))$ □

Example. The following are not propositions:

$F\ F$, $(b \vee (c)$, $(b) \wedge)$, $a+b$ □

As seen in the above syntax, five operators are defined over values of type *Boolean*:

negation:	(**not** b), or $(\neg b)$
conjunction:	(b **and** c), or $(b \wedge c)$
disjunction:	(b **or** c), or $(b \vee c)$
implication:	(b **imp** c), or $(b \Rightarrow c)$
equality:	(b **equals** c), or $(b = c)$

Two different notations have been given for each operator, a name and a mathematical symbol. The name indicates how to pronounce it, and its use also makes typing easier when a typewriter does not have the corresponding mathematical symbol.

The following terminology is used. $(b \wedge c)$ is called a *conjunction*; its operands b and c called *conjuncts*. $(b \vee c)$ is called a *disjunction*; its operands b and c are called *disjuncts*. $(b \Rightarrow c)$ is called an *implication*; its *antecedent* is b and its *consequent* is c.

1.2 Evaluation of Constant Propositions

Thus far we have given a *syntax* for propositions; we have defined the set of well-formed propositions. We now give a semantics (meaning) by showing how to evaluate them.

We begin by defining evaluation of *constant propositions* —propositions that contain only constants as operands— and we do this in three cases based on the structure of a proposition e: for e with no operators, for e with one operator, and for e with more than one operator.

(1.2.1) **Case 1.** The value of proposition T is T; the value of F is F.

(1.2.2) **Case 2.** The values of $(\neg b)$, $(b \wedge c)$, $(b \vee c)$, $(b \Rightarrow c)$ and $(b = c)$, where b and c are each one of the constants T and F, are given by the following table (called a *truth table*). Each row of the table contains possible values for the operands b and c and, for these values, shows the value of each of the five operations. For example, from the last row we see that the value of $(\neg T)$ is F and that the values of $(T \wedge T)$, $(T \vee T)$, $(T \Rightarrow T)$ and $(T = T)$ are all T.

(1.2.3)

b	c	$(\neg b)$	$(b \wedge c)$	$(b \vee c)$	$(b \Rightarrow c)$	$(b = c)$
F	F	T	F	F	T	T
F	T	T	F	T	T	F
T	F	F	F	T	F	F
T	T	F	T	T	T	T

(1.2.4) **Case 3.** The value of a constant proposition with more than one operator is found by repeatedly applying (1.2.2) to a subproposition of the constant proposition and replacing the subproposition by its value, until the proposition is reduced to T or F.

We give an example of evaluation of a proposition:

$$\begin{aligned} &((T \wedge T) \Rightarrow F) \\ =\ &(T \Rightarrow F) \\ =\ &F \end{aligned}$$

Remark: The description of the operations in terms of a truth table, which lists *all* possible operand combinations and their values, can be given only because the set of possible values is finite. For example, no such table could be given for operations on integers. □

The names of the operations correspond fairly closely to their meanings in English. For example, "not true" usually means "false", and "not false" "true". But note that operation **or** denotes "inclusive or" and not "exclusive or". That is, $(T \vee T)$ is T, while the "exclusive or" of T and T is false.

Also, there is no causality implied by operation **imp**. The sentence "If it rains, the picnic is cancelled" can be written in propositional form as $(rain \Rightarrow no\ picnic)$. From the English sentence we infer that the lack of rain means there will be a picnic, but *no* such inference can be made from the proposition $(rain \Rightarrow no\ picnic)$.

1.3 Evaluation of Propositions in a State

A proposition like $((\neg c) \vee d)$ can appear in a program in several places, for example in an assignment statement $b := ((\neg c) \vee d)$ and in an **if**-statement **if** $((\neg c) \vee d)$ **then** $\cdots$. When the statement in which the proposition appears is to be executed, the proposition is evaluated in the current machine "state" to produce either T or F. To define this evaluation requires a careful explanation of the notion of "state".

A state associates identifiers with values. For example, in state s (say), identifier c could be associated with value F and identifier d with T. In terms of a computer memory, when the computer is in state s, locations named c and d contain the values F and T, respectively. In another state, the associations could be (c, T) and (d, F). The crucial point here is that a state consists of a set of pairs (identifier, value) in which all the identifiers are distinct, i.e. the state is a function:

(1.3.1) **Definition.** A *state* s is a function from a set of identifiers to the set of values T and F. □

Example. Let state s be the function defined by the set $\{(a, T), (bc, F), (yl, T)\}$. Then $s(a)$ denotes the value determined by applying state (function) s to identifier a: $s(a) = T$. Similarly, $s(bc) = F$ and $s(yl) = T$. □

(1.3.2) **Definition.** Proposition e is *well-defined in state* s if each identifier in e is associated with either T or F in state s. □

In state $s = \{(b, T), (c, F)\}$, proposition $(b \vee c)$ is well-defined while proposition $(b \vee d)$ is not.

Let us now extend the notation $s(identifier)$ to define the value of a proposition in a state. For any state s and proposition e, $s(e)$ will denote the value resulting from evaluating e in state s. Since an identifier b is also a proposition, we will be careful to make sure that $s(b)$ will still denote the value of b in state s.

(1.3.3) **Definition**. Let proposition e be well-defined in state s. Then $s(e)$, the value of e in state s, is the value obtained by replacing all occurrences of identifiers b in e by their values $s(b)$ and evaluating the resulting constant proposition according to the rules given in the previous section 1.2. □

Example. $s(((\neg b) \vee c))$ is evaluated in state $s = \{(b, T), (c, F)\}$:

$$\begin{aligned} & s(((\neg b) \vee c)) \\ & \quad = ((\neg T) \vee F) \quad (b \text{ has been replaced by } T, c \text{ by } F) \\ & \quad = (F \vee F) \\ & \quad = F \quad \square \end{aligned}$$

1.4 Precedence Rules for Operators

The previous sections dealt with a restricted form of propositions, so that evaluation of propositions could be explained without having to deal with the precedence of operators. We now relax this restriction.

Parentheses can be omitted or included at will around any proposition. For example, the proposition $((b \vee c) \Rightarrow d)$ can be written as $b \vee c \Rightarrow d$. In this case, additional rules define the order of evaluation of subpropositions. These rules, which are similar to those for arithmetic expressions are:

1. Sequences of the same operator are evaluated from left to right, e.g. $b \wedge c \wedge d$ is equivalent to $((b \wedge c) \wedge d)$.
2. The order of evaluation of different, adjacent operators is given by the list: **not** (has highest precedence and binds tightest), **and, or, imp, equals.**

It is usually better to make liberal use of parentheses in order to make the order of evaluation clear, and we will usually do so.

Examples	$\neg b = b \wedge c$	is equivalent to	$(\neg b) = (b \wedge c)$
	$b \vee \neg c \Rightarrow d$	is equivalent to	$(b \vee (\neg c)) \Rightarrow d$
	$b \Rightarrow c \Rightarrow d \wedge e$	is equivalent to	$(b \Rightarrow c) \Rightarrow (d \wedge e)$ □

The following BNF grammar defines the syntax of propositions, giving enough structure so that precedences can be deduced from it. (The non-terminal <identifier> has been left undefined and has its usual meaning).

1.	<proposition>	::=	<imp-expr>
2.		|	<proposition> = <imp-expr>
3.	<imp-expr>	::=	<expr>
4.		|	<imp-expr> $\Rightarrow$ <expr>
5.	<expr>	::=	<term>
6.		|	<expr> $\vee$ <term>
7.	<term>	::=	<factor>
8.		|	<term> $\wedge$ <factor>
9.	<factor>	::=	$\neg$ <factor>
10.		|	(<proposition>)
11.		|	T
12.		|	F
13.		|	<identifier>

We now define $s(e)$, the value of proposition e in state s, recursively, based on the structure of e given by the grammar. That is, for each rule of the grammar, we show how to evaluate e if it has the form given by that rule. For example, rule 6 indicates that for an <expr> of the form <expr> $\vee$ <term>, its value is the value found by applying operation **or** to the values s(<expr>) and s(<term>) of its operands <expr> and <term>. The values of the five operations $=$, $\Rightarrow$, $\vee$, $\wedge$ and $\neg$ used in rules 2, 4, 6, 8 and 9 are given by truth table (1.2.3).

1.	s(<proposition>)	=	s(<imp-expr>)
2.	s(<proposition>)	=	s(<proposition>) = s(<imp-expr>)
3.	s(<imp-expr>)	=	s(<expr>)
4.	s(<imp-expr>)	=	s(<imp-expr>) $\Rightarrow$ s(<expr>)
5.	s(<expr>)	=	s(<term>)
6.	s(<expr>)	=	s(<expr>) $\vee$ s(<term>)
7.	s(<term>)	=	s(<factor>)
8.	s(<term>)	=	s(<term>) $\wedge$ s(<factor>)
9.	s(<factor>)	=	$\neg$ s(<factor>)
10.	s(<factor>)	=	s(<proposition>)
11.	s(<factor>)	=	T
12.	s(<factor>)	=	F
13.	s(<factor>)	=	s(<identifier>) (the value of <identifier> in s)

An example of evaluation using a truth table

Let us compute values of the proposition $(b \Rightarrow c) = (\neg b \vee c)$ for all possible operand values using a truth table. In the table below, each row gives possible values for b and c and the corresponding values of $\neg b$, $\neg b \vee c$, $b \Rightarrow c$ and the final proposition. This truth table shows how one builds a truth table for a proposition, by beginning with the values of the identifiers, then showing the values of the smallest subpropositions, then the next smallest, and building up to the complete proposition.

As can be seen, the values of $\neg b \vee c$ and $b \Rightarrow c$ are the same in each state, and hence the propositions are equivalent and can be used interchangeably. In fact, one often finds $b \Rightarrow c$ *defined* as $\neg b \vee c$. Similarly, $b = c$ is often defined as an abbreviation for $(b \Rightarrow c) \wedge (c \Rightarrow b)$ (see exercise 2i).

b	c	$\neg b$	$\neg b \vee c$	$b \Rightarrow c$	$(b \Rightarrow c) = (\neg b \vee c)$
F	F	T	T	T	T
F	T	T	T	T	T
T	F	F	F	F	T
T	T	F	T	T	T

1.5 Tautologies

A *Tautology* is a proposition that is true in every state in which it is well-defined. For example, proposition T is a tautology and F is not. The proposition $b \vee \neg b$ is a tautology, as can be seen by evaluating it with $b = T$ and $b = F$:

$$\begin{aligned} T \vee \neg T &= T \vee F = T \\ F \vee \neg F &= F \vee T = T \end{aligned}$$

or, in truth-table form:

b	$\neg b$	$b \vee \neg b$
T	F	T
F	T	T

The basic way to show that a proposition is a tautology is to show that its evaluation yields T in every possible state. Unfortunately, each extra identifier in a proposition doubles the number of combinations of values for identifiers —for a proposition with i distinct identifiers there are 2^i cases! Hence, the work involved can become tedious and time consuming. To illustrate this, (1.5.1) contains the truth table for proposition $(b \wedge c \wedge d) \Rightarrow (d \Rightarrow b)$, which has three distinct identifiers. By taking some shortcuts, the work can be reduced. For example, a glance at truth table (1.2.3) indicates that operation **imp** is true whenever its antecedent is false, so that its consequent need only be evaluated if its antecedent is true. In example (1.5.1) there is only one state in which the antecedent $b \wedge c \wedge d$ is true —the state in which b, c and d are true— and hence we need only the top line of truth table (1.5.1).

(1.5.1)

b c d	$b \wedge c \wedge d$	$d \Rightarrow b$	$(b \wedge c \wedge d) \Rightarrow (d \Rightarrow b)$
T T T	T	T	T
T T F	F	T	T
T F T	F	T	T
T F F	F	T	T
F T T	F	F	T
F T F	F	T	T
F F T	F	F	T
F F F	F	T	T

Using such informal reasoning helps reduce the number of states in which the proposition must be evaluated. Nevertheless, the more distinct identifiers a proposition has the more states to inspect, and evaluation soon becomes infeasible. Later chapters investigate other methods for proving that a proposition is a tautology.

Disproving a conjecture

Sometimes we conjecture that a proposition e is a tautology, but are unable to develop a proof of it, so we decide to try to disprove it. What does it take to disprove such a conjecture?

It may be possible to prove the converse —i.e. that $\neg e$ is a tautology— but the chances are slim. If we had reason to believe a conjecture, it is unlikely that its converse is true. Much more likely is that it is true in most states but false in one or two, and to disprove it we need only find one such state:

To prove a conjecture, it is necessary to prove that it is true in all cases; to disprove a conjecture, it is sufficient to find a single case where it is false.

1.6 Propositions as Sets of States

A proposition *represents*, or describes, the set of states in which it is true. Conversely, for any set of states containing only identifiers associated with T or F we can derive a proposition that represents that state set. Thus, the empty set, the set containing no states, is represented by proposition F because F is true in no state. The set of all states is represented by proposition T because T is true in all states. The following example illustrates how one can derive a proposition that represents a given set of states. The resulting proposition contains only the operators **and**, **or** and **not**.

Example. The set of two states $\{(b, T), (c, T), (d, T)\}$ and $\{(b, F), (c, T), (d, F)\}$, is represented by the proposition

$$(b \wedge c \wedge d) \vee (\neg b \wedge c \wedge \neg d) \quad \square$$

The connection between a proposition and the set of states it represents is so strong that we often *identify* the two concepts. Thus, instead of writing "the set of states in which $b \vee \neg c$ is true" we may write "the states in $b \vee \neg c$". Though it is a sloppy use of English, it is at times convenient.

In connection with this discussion, the following terminology is introduced. Proposition b is *weaker* than c if $c \Rightarrow b$. Correspondingly, c is said to be *stronger* than b. A stronger proposition makes more restrictions on the combinations of values its identifiers can be associated with, a weaker proposition makes fewer. In terms of sets of states, b is as weak as c if it is "less restrictive": if b's set of states includes at least c's states, and possibly more. The weakest proposition is T (or any tautology), because it represents the set of all states; the strongest is F, because it represents the set of no states.

1.7 Transforming English to Propositional Form

At this point, we translate a few sentences into propositional form. Consider the sentence "If it rains, the picnic is cancelled." Let identifier r stand for the proposition "it rains" and let identifier pc represent "the picnic is cancelled". Then the sentence can be written as $r \Rightarrow pc$.

As shown by this example, the technique is to represent "atomic parts" of a sentence —how these are chosen is up to the translator— by identifiers and to describe their relationship using Boolean operators. Here are some more examples, using identifiers r, pc, wet, and s defined as follows:

it rains: r
picnic is cancelled: pc
be wet: wet
stay at home: s

1. If it rains but I stay at home, I won't be wet: $(r \wedge s) \Rightarrow \neg wet$
2. I'll be wet if it rains: $r \Rightarrow wet$
3. If it rains and the picnic is not cancelled or I don't stay home, I'll be wet: Either $((r \wedge \neg pc) \vee \neg s) \Rightarrow wet$ or $((r \wedge (\neg pc \vee \neg s) \Rightarrow wet$. The English is ambiguous; the latter proposition is probably the desired one.
4. Whether or not the picnic is cancelled, I'm staying home if it rains: $(pc \vee \neg pc) \wedge r \Rightarrow s$. This reduces to $r \Rightarrow s$.
5. Either it doesn't rain or I'm staying home: $\neg r \vee s$.

Exercises for Chapter 1

1. Each line contains a proposition and two states $s1$ and $s2$. Evaluate the proposition in both states.

	proposition	state $s1$				state $s2$			
		m	n	p	q	m	n	p	q
(a)	$\neg(m \vee n)$	T	F	T	T	F	T	T	T
(b)	$\neg m \vee n$	T	F	T	T	F	T	T	T
(c)	$\neg(m \wedge n)$	T	F	T	T	F	T	T	T
(d)	$\neg m \wedge n$	T	F	T	T	F		T	
(e)	$(m \vee n) \Rightarrow p$	T	F	T	T	T	T	F	T
(f)	$m \vee (n \Rightarrow p)$	T	F	T	T	T	T	F	T
(g)	$(m = n) \wedge (p = q)$	F	F	T	F	T	F	T	F
(h)	$m = (n \wedge (p = q))$	F	F	T	F	T	F	T	F
(i)	$m = (n \wedge p = q)$	F	F	T	F	T	F	T	F
(j)	$(m = n) \wedge (p \Rightarrow q)$	F	T	F	T	T	T	F	F
(k)	$(m = n \wedge p) \Rightarrow q$	F	T	F	T	T	T	F	F
(l)	$(m \Rightarrow n) \Rightarrow (p \Rightarrow q)$	F	F	F	F	T	T	T	T
(m)	$(m \Rightarrow (n \Rightarrow p)) \Rightarrow q$	F	F	F	F	T	T	T	T

2. Write truth tables to show the values of the following propositions in all states:

(a)	$b \vee c \vee d$	(e)	$\neg b \Rightarrow (b \vee c)$
(b)	$b \wedge c \wedge d$	(f)	$\neg b = (b \vee c)$
(c)	$b \wedge (c \vee d)$	(g)	$(\neg b = c) \vee b$
(d)	$b \vee (c \wedge d)$	(h)	$(b \vee c) \wedge (b \Rightarrow c) \wedge (c \Rightarrow b)$
		(i)	$(b = c) = (b \Rightarrow c) \wedge (c \Rightarrow b)$

3. Translate the following sentences into propositional form.

(a) $x < y$ or $x = y$.
(b) Either $x < y$, $x = y$, or $x > y$.
(c) If $x > y$ and $y > z$, then $v = w$.
(d) The following are all true: $x < y$, $y < z$ and $v = w$.
(e) At most one of the following is true: $x < y$, $y < z$ and $v = w$.
(f) None of the following are true: $x < y$, $y < z$ and $v = w$.
(g) The following are not all true at the same time: $x < y$, $y < z$ and $v = w$.
(h) When $x < y$, then $y < z$; when $x \geqslant y$, then $v = w$.
(i) When $x < y$ then $y < z$ means that $v = w$, but if $x \geqslant y$ then $y < z$ doesn't hold; however, if $v = w$ then $x < y$.
(j) If execution of program P is begun with $x < y$, then execution terminates with $y = 2^x$.
(k) Execution of program P begun with $x < 0$ will not terminate.

4. Below are some English sentences. Introduce identifiers to represent the simple ones (e.g. "it's raining cats and dogs.") and then translate the sentences into propositions.

(a) Whether or not it's raining, I'm going swimming.
(b) If it's raining I'm not going swimming.
(c) It's raining cats and dogs.
(d) It's raining cats or dogs.
(e) If it rains cats and dogs I'll eat my hat, but I won't go swimming.
(f) If it rains cats and dogs while I am swimming I'll eat my hat.

Chapter 2
Reasoning using Equivalence Transformations

Evaluating propositions is rarely our main task. More often we wish to manipulate them in some manner in order to derive "equivalent" but simpler ones (easier to read and understand). Two propositions (or, in general, expressions) are equivalent if they have the same value in every state. For example, since $a+(c-a)=c$ is always true for integer variables a and c, the two integer expressions $a+(c-a)$ and c are equivalent, and $a+(c-a)=c$ is called an equivalence.

This chapter defines equivalence of propositions in terms of the evaluation model of chapter 1. A list of useful equivalences is given, together with two rules for generating others. The idea of a "calculus" is discussed, and the rules are put in the form of a formal calculus for "reasoning" about propositions.

These rules form the basis for much of the manipulations we do with propositions and are very important for later work on developing programs. The chapter should be studied carefully.

2.1 The Laws of Equivalence

For propositions, we define equivalence in terms of operation **equals** and the notion of a tautology as follows:

(2.1.1) **Definition**. Propositions *E1* and *E2* are *equivalent iff* $E1 = E2$ is a tautology. In this case, $E1 = E2$ is an *equivalence.* □

Thus, an equivalence is an equality that is a tautology.

Below, we give a list of equivalences; these are the basic equivalences from which all others will be derived, so we call them the *laws of equivalence.* Actually, they are "schemas": the identifiers *E1*, *E2* and *E3* within them are parameters, and one arrives at a particular equivalence by substituting particular propositions for them. For example, substituting $x \vee y$ for $E1$ and z for $E2$ in the first law of Commutativity, $(E1 \wedge E2) = (E2 \wedge E1)$, yields the equivalence

$$((x \vee y) \wedge z) = (z \wedge (x \vee y))$$

Remark: Parentheses are inserted where necessary when performing a substitution so that the order of evaluation remains consistent with the original proposition. For example, the result of substituting $x \vee y$ for b in $b \wedge z$ is $(x \vee y) \wedge z$, and not $x \vee y \wedge z$, which is equivalent to $x \vee (y \wedge z)$. □

1. **Commutative Laws** (These allow us to reorder the operands of **and, or** and **equality**):
   $(E1 \wedge E2) \;=\; (E2 \wedge E1)$
   $(E1 \vee E2) \;=\; (E2 \vee E1)$
   $(E1 = E2) \;=\; (E2 = E1)$

2. **Associative Laws** (These allow us to dispense with parentheses when dealing with sequences of **and** and sequences of **or**):
   $E1 \wedge (E2 \wedge E3) \;=\; (E1 \wedge E2) \wedge E3$ (so write both as $E1 \wedge E2 \wedge E3$)
   $E1 \vee (E2 \vee E3) \;=\; (E1 \vee E2) \vee E3$

3. **Distributive Laws** (These are useful in factoring a proposition, in the same way that we rewrite 2*(3+4) as (2*3)+(2*4)):
   $E1 \vee (E2 \wedge E3) \;=\; (E1 \vee E2) \wedge (E1 \vee E3)$
   $E1 \wedge (E2 \vee E3) \;=\; (E1 \wedge E2) \vee (E1 \wedge E3)$

4. **De Morgan's Laws** (After Augustus De Morgan, a 19th century English mathematician who, along with Boole, laid much of the foundations for mathematical logic):
   $\neg(E1 \wedge E2) \;=\; \neg E1 \vee \neg E2$
   $\neg(E1 \vee E2) \;=\; \neg E1 \wedge \neg E2$

5. **Law of Negation**: $\neg(\neg E1) \;=\; E1$

6. **Law of the Excluded Middle**: $E1 \vee \neg E1 \;=\; T$

7. **Law of Contradiction**: $E1 \wedge \neg E1 \;=\; F$

8. **Law of Implication**: $E1 \Rightarrow E2 \;=\; \neg E1 \vee E2$

9. **Law of Equality**: $(E1 = E2) \;=\; (E1 \Rightarrow E2) \wedge (E2 \Rightarrow E1)$

10. **Laws of or-simplification**:
    $E1 \vee E1 \;=\; E1$
    $E1 \vee T \;=\; T$
    $E1 \vee F \;=\; E1$
    $E1 \vee (E1 \wedge E2) \;=\; E1$

11. **Laws of and-simplification**:
    $E1 \wedge E1 \;=\; E1$
    $E1 \wedge T \;=\; E1$
    $E1 \wedge F \;=\; F$
    $E1 \wedge (E1 \vee E2) \;=\; E1$

12. **Law of Identity**: $E1 = E1$

Don't be alarmed at the number of laws. Most of them you have used many times, perhaps unknowingly, and this list will only serve to make you more aware of them. Study the laws carefully, for they are used over and over again in manipulating propositions. Do some of the exercises at the end of this section until the use of these laws becomes second nature. Knowing the laws by name makes discussions of their use easier.

The law of the Excluded Middle deserves some comment. It means that either b or $\neg b$ must be true in any state; there can be no middle ground. Some don't believe this law, at least in all its generality. In fact, here is a counterexample to it, in English. Consider the sentence

This sentence is false.

which we might consider as the meaning of an identifier b. Is it true or false? It can't be true, because it says it is false; it can't be false, because then it would be true! The sentence is neither true nor false, and hence violates the law of the Excluded Middle. The paradox arises because of the self-referential aspect of the sentence —it indicates something about itself, as do all paradoxes. [Here is another paradox to ponder: a barber in a small town cuts the hair of every person in town except for those who cut their own. Who cuts the barber's hair?] In our formal system, there will be no way to introduce such self-referential treatment, and the law of the Excluded Middle holds. But this means we cannot express *all* our thoughts and arguments in the formal system.

Finally, the laws of Equality and Implication deserve special mention. Together, they define **equality** and **imp** in terms of other operators: $b = c$ can always be replaced by $(b \Rightarrow c) \wedge (c \Rightarrow b)$ and $\neg b \Rightarrow c$ by $b \vee c$. This reinforces what we said about the two operations in chapter 1.

Proving that the logical laws are equivalences

We have stated, without proof, that laws 1-12 are equivalences. One way to prove this is to build truth tables and note that the laws are true in all states. For example, the first of De Morgan's laws, $\neg(E1 \wedge E2) = \neg E1 \vee \neg E2$, has the following truth table:

$E1$	$E2$	$E1 \wedge E2$	$\neg(E1 \wedge E2)$	$\neg E1$	$\neg E2$	$\neg E1 \vee \neg E2$	$\neg(E1 \wedge E2) = \neg E1 \vee \neg E2$
F	F	F	T	T	T	T	T
F	T	F	T	T	F	T	T
T	F	F	T	F	T	T	T
T	T	T	F	F	F	F	T

Clearly, the law is true in all states (in which it is well-defined), so that it is a tautology.

Exercise 1 concerns proving all the laws to be equivalences.

2.2 The Rules of Substitution and Transitivity

Thus far, we have just discussed some basic equivalences. We now turn to ways of generating other equivalences, without having to check their truth tables. One rule we all use in transforming expressions, usually without explicit mention, is the rule of "substitution of equals for equals". Here is an example of the use of this rule. Since $a+(c-a)=c$, we can substitute for expression $a+(c-a)$ in $(a+(c-a))*d$ to conclude that $(a+(c-a))*d = c*d$; we simply replace $a+(c-a)$ in $(a+(c-a))*d$ by the simpler, but equivalent, expression c.

The rule of substitution is:

(2.2.1) **Rule of Substitution.** Let $e1=e2$ be an equivalence and $E(p)$ be a proposition, written as a function of one of its identifiers p. Then $E(e1)=E(e2)$ and $E(e2)=E(e1)$ are also equivalences. □

Here is an example of the use of the rule of Substitution. The law of Implication indicates that $(b \Rightarrow c) = (\neg b \vee c)$ is an equivalence. Consider the proposition $E(p) = d \vee p$. With

$$e1 = b \Rightarrow c \text{ and}$$
$$e2 = \neg b \vee c$$

we have

$$E(e1) = d \vee (b \Rightarrow c)$$
$$E(e2) = d \vee (\neg b \vee c)$$

so that $(d \vee (b \Rightarrow c) = d \vee (\neg b \vee c)$ is an equivalence.

In using the rule of Substitution, we often use the following form. The proposition that we conclude is an equivalence is written on one line. The initial proposition appears to the left of the equality sign and the one that results from the substitution appears to the right, followed by the name of the law $e1 = e2$ used in the application:

$$d \vee (b \Rightarrow c) = d \vee (\neg b \vee c) \quad \text{(Implication)}$$

We need one more rule for generating equivalences:

(2.2.2) **Rule of Transitivity.** If $e1 = e2$ and $e2 = e3$ are equivalences, then so is $e1 = e3$ (and hence $e1$ is equivalent to $e3$). □

Example. We show that $(b \Rightarrow c) = (\neg c \Rightarrow \neg b)$ is an equivalence (an explanation of the format follows):

$$\begin{array}{ll} b \Rightarrow c & \\ = \neg b \vee c & \text{(Implication)} \\ = c \vee \neg b & \text{(Commutativity)} \\ = \neg\neg c \vee \neg b & \text{(Negation)} \\ = \neg c \Rightarrow \neg b & \text{(Implication)} \end{array}$$

This is read as follows. First, lines 1 and 2 indicate that $b \Rightarrow c$ is equivalent to $\neg b \vee c$, by virtue of the rule of Substitution and the law of Implication. Secondly, lines 2 and 3 indicate that $(\neg b \vee c)$ is equivalent to $c \vee \neg b$, by virtue of the rule of Substitution and the law of Commutativity. We also conclude, using the rule of Transitivity, that the first proposition, $b \Rightarrow c$, is equivalent to the third, $c \vee \neg b$. Continuing in this fashion, each pair of lines gives an equivalence and the reasons why the equivalence holds. We finally conclude that the first proposition, $b \Rightarrow c$, is equivalent to the last, $\neg c \Rightarrow \neg b$. □

Example. We show that the law of Contradiction can be proved from the others. The portion of each proposition to be replaced in each step is underlined in order to make it easier to identify the substitution.

$$\begin{array}{rll} \underline{\neg(b \wedge \neg b)} & = \neg b \vee \underline{\neg\neg b} & \text{(De Morgan's Law)} \\ & = \underline{\neg b \vee b} & \text{(Negation)} \\ & = \underline{b \vee \neg b} & \text{(Commutativity)} \\ & = T & \text{(Excluded Middle)} \quad \square \end{array}$$

Generally speaking, such fine detail is unnecessary. The laws of Commutativity and Associativity are often used without explanation, and the application of several steps can appear on one line. For example:

$$\begin{array}{ll} (b \wedge (b \Rightarrow c)) \Rightarrow c & \\ \quad = \neg(b \wedge (\neg b \vee c)) \vee c & \text{(Implication, 2 times)} \\ \quad = \neg b \vee \neg(\neg b \vee c) \vee c & \text{(De Morgan)} \\ \quad = T & \text{(Excluded Middle)} \end{array}$$

Transforming an implication

Suppose we want to prove that

(2.2.3) $E1 \wedge E2 \wedge E3 \Rightarrow E$

is an equivalence. The proposition is transformed as follows:

$$\begin{array}{ll} (E1 \wedge E2 \wedge E3) \Rightarrow E & \\ \quad = \neg(E1 \wedge E2 \wedge E3) \vee E & \text{(Implication)} \\ \quad = \neg E1 \vee \neg E2 \vee \neg E3 \vee E & \text{(De Morgan)} \end{array}$$

The final proposition is true in any state in which at least one of $\neg E1$, $\neg E2$, $\neg E3$ and E is true. Hence, to prove that (2.2.3) is a tautology we need only prove that in any state in which three of them are false the fourth is true. And we can choose which three to assume false, based on their form, in order to develop the simplest proof.

With an argument similar to the one just given, we can see that the five statements

$$\begin{array}{l} E1 \wedge E2 \wedge E3 \Rightarrow E \\ E1 \wedge E2 \wedge \neg E \Rightarrow \neg E3 \\ E1 \wedge \neg E \wedge E3 \Rightarrow \neg E2 \\ \neg E \wedge E2 \wedge E3 \Rightarrow \neg E1 \end{array}$$

(2.2.4) $\neg E1 \vee \neg E2 \vee \neg E3 \vee E$

are equivalent and we can choose which to work with. When given a proposition like (2.2.3), eliminating implication completely in favor of disjunctions like (2.2.4) can be helpful. Likewise, when formulating a problem, put it in the form of a disjunction right from the beginning.

Example. Prove that

$$(\neg(b \Rightarrow c) \wedge \neg(\neg b \Rightarrow (c \vee d))) \Rightarrow (\neg c \Rightarrow d)$$

is a tautology. Eliminate the main implication and use De Morgan's law:

$$\neg\neg(b \Rightarrow c) \vee \neg\neg(\neg b \Rightarrow (c \vee d)) \vee (\neg c \Rightarrow d)$$

Now simplify using Negation and eliminate the other implications:

$$(\neg b \vee c) \vee (b \vee c \vee d) \vee (c \vee d)$$

Use the laws of Associativity, Commutativity and **or**-simplification to arrive at

$$b \vee \neg b \vee c \vee d$$

which is true because of the laws of the Excluded Middle, $b \vee \neg b = T$, and **or**-simplification. This problem, which at first looked quite difficult, became simple when the implications were eliminated.

2.3 A Formal System of Axioms and Inference Rules

A *calculus*, according to Webster's Third International Dictionary, is a method or process of reasoning by computation of symbols. In section 2.2 we presented a calculus, for by performing some symbol manipulation according to rules of Substitution and Transitivity we can reason with propositions. For obvious reasons, the system presented here is called a *propositional calculus*.

We are careful to say *a* propositional calculus, and not *the* propositional calculus. With slight changes in the rules we can have a different calculus. Or we can invent a completely different set of rules and a completely different calculus, which is better suited for other purposes.

We want to emphasize the nature of this calculus as a formal system for manipulating propositions. To do this, let us put aside momentarily the notions of state and evaluation and see whether equivalences, which we will call *theorems*, can be discussed without them. First, define the propositions that arise directly from laws 1-12 to be *theorems*. They are also called *axioms* (and the laws 1-12 are *axiom schemas*), because their *theorem*hood is taken at face value, without proof.

(2.3.1) **Axioms.** Any proposition that arises by substituting propositions for *E1*, *E2* and *E3* in one of the Laws 1-12 is called a *theorem*. □

Next, define the propositions that arise by using the rules of Substitution and Transitivity and an already-derived *theorem* to be a *theorem*. In this context, the rules are often called *inference rules*, for they can be used to *infer* that a proposition is a *theorem*. An inference rule is often written in the form

$$\frac{E_1, \cdots, E_n}{E} \quad \text{and} \quad \frac{E_1, E_2, \cdots, E_n}{E, E_0}$$

where the E_i and E stand for arbitrary propositions. The inference rule has the following meaning. If propositions $E_1, \cdots, E_n$ are *theorems*, then so is proposition E (and E_0 in the second case). Written in this form, the rules of Substitution and Transitivity are

(2.3.2) **Rule of Substitution:** $$\frac{e1 = e2}{E(e1) = E(e2),\ E(e2) = E(e1)}$$

(2.3.3) **Rule of Transitivity:** $$\frac{e1 = e2,\ e2 = e3}{e1 = e3}$$

A *theorem* of the formal system, then, is either an axiom (according to (2.3.1) or a proposition that is derived from one of the inference rules (2.3.2) and (2.3.3).

Note carefully that this is a totally different system for dealing with propositions, which has been defined without regard to the notions of states and evaluation. The syntax of propositions is the same, but what we do with propositions is entirely different. Of course, there is a relation between the formal system and the system of evaluation given in the previous chapter. Exercises 9 and 10 call for proof of the following relationship: for any tautology e in the sense of chapter 1, $e = T$ is a *theorem*, and vice versa.

Exercises for Chapter 2

1. Verify that laws 1-12 are equivalences by building truth tables for them.

2. Prove the law of Identity, $e = e$, using the rules of Substitution and Transitivity and the laws 1-11.

3. Prove that $\neg T = F$ is an equivalence, using the rules of Substitution and Transitivity and the laws 1-12.

4. Prove that $\neg F = T$ is an equivalence, using the rules of Substitution and Transitivity and the laws 1-12.

5. Each column below consists of a sequence of propositions, each of which (except the first) is equivalent to its predecessor. The equivalence can be shown by one application of the rule of Substitution and one of the laws 1-12 or the results of exercises 3-4. Identify the law (as is done for the first two cases).

(a) $(x \wedge y) \vee (z \wedge \neg z)$	(a) $\neg(\neg b \wedge (\neg b \Rightarrow z)) \vee z$
(b) $(x \wedge y) \vee F$ Contradiction	(b) $(\neg b \wedge (\neg b \Rightarrow z)) \Rightarrow z)$
(c) $x \wedge y$ or-simplification	(c) $(\neg b \wedge (\neg\neg b \vee z)) \Rightarrow z$
(d) $(x \wedge y) \vee F$	(d) $(\neg b \wedge (\neg\neg b \vee \neg\neg z)) \Rightarrow z$
(e) $(x \wedge y) \vee (F \wedge z)$	(e) $(\neg b \wedge \neg(\neg b \wedge \neg z)) \Rightarrow z$
(f) $(x \wedge y) \vee (F \wedge z)$	(f) $(\neg b \wedge \neg(\neg b \wedge \neg z)) \Rightarrow z$
(g) $(x \wedge y) \vee ((x \wedge \neg x) \wedge z)$	(g) $\neg(b \vee (\neg b \wedge \neg z)) \Rightarrow z$

(h) $(x \wedge y) \vee (x \wedge (\neg x \wedge z))$	(h) $\neg((b \vee \neg b) \wedge (b \vee \neg z)) \Rightarrow z$
(i) $x \wedge (y \vee (\neg x \wedge z))$	(i) $\neg(T \wedge (b \vee \neg z)) \Rightarrow z$
(j) $x \wedge (y \vee \neg x) \wedge (y \vee z)$	(j) $\neg(b \vee \neg z) \Rightarrow z$
(k) $x \wedge (\neg x \vee y) \wedge (z \vee y)$	(k) $\neg\neg(b \vee \neg z) \vee z$
(l) $x \wedge (\neg x \vee \neg\neg y) \wedge (z \vee y)$	(l) $(b \vee \neg z) \vee z)$
(m) $x \wedge \neg(x \wedge \neg y) \wedge (z \vee y)$	(m) $b \vee (\neg z \vee z)$

6. Each proposition below can be simplified to one of the six propositions F, T, x, y, $x \wedge y$, and $x \vee y$. Simplify them, using the rules of Substitution and Transitivity and the laws 1-12.

(a) $x \vee (y \vee x) \vee \neg y$	(g) $\neg x \Rightarrow (x \wedge y)$
(b) $(x \vee y) \wedge (x \vee \neg y)$	(h) $T \Rightarrow (\neg x \Rightarrow x)$
(c) $x \vee y \vee \neg x$	(i) $x \Rightarrow (y \Rightarrow (x \wedge y))$
(d) $(x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y)$	(j) $\neg x \Rightarrow (\neg x \Rightarrow (\neg x \wedge y))$
(e) $(x \wedge y) \vee (x \wedge \neg y) \vee (\neg x \wedge y) \vee (\neg x \wedge \neg y)$	(k) $\neg y \Rightarrow y$
(f) $(\neg x \wedge y) \vee x$	(l) $\neg y \Rightarrow \neg y$

7. Show that any proposition e can be transformed into an equivalent proposition in *disjunctive normal form* —i.e. one that has the form

$$e_0 \vee \cdots \vee e_n \text{ where each } e_i \text{ has the form } g_0 \wedge \cdots \wedge g_m$$

Each g_j is an identifier *id*, a unary operator $\neg id$, T or F. Furthermore, the identifiers in each e_i are distinct.

8. Show that any proposition e can be transformed into an equivalent proposition in *conjunctive normal form* —i.e. one that has the form

$$e_0 \wedge \cdots \wedge e_n \text{ where each } e_i \text{ has the form } g_0 \vee \cdots \vee g_m$$

Each g_j is an identifier *id*, a unary operator $\neg id$, T or F. Furthermore, the identifiers in each e_i are distinct.

9. Prove that any *theorem* generated using laws 1-12 and the rules of Substitution and Transitivity is a tautology, by proving that laws 1-12 are tautologies (see exercise 1) and showing that the two rules can generate only tautologies.

10. Prove that if e is a tautology, then $e = T$ can be proved to be an equivalence using only the laws 1-12 and the rules of Substitution and Transitivity. Hint: use exercise 8.

Chapter 3
A Natural Deduction System

This chapter introduces another formal system of axioms and inference rules for deducing proofs that propositions are tautologies. It is called a "natural deduction system" because it is meant to mimic the patterns of reasoning that we "naturally" use in making arguments in English.

This material is not used in later parts of the book, and can be skipped. The equivalence transformation system discussed in chapter 2 serves more than adequately in developing correct programs later on. One could go further and say that the equivalence transformation system is more suited to our needs, although, this may be a matter of taste.

Nevertheless, study of this chapter is worthwhile for several reasons. The formal system presented here is minimal: there are *no* axioms and a minimal number of inference rules. Thus, one can see what it takes to start with a bare-bones system and build up enough theorems to the point where further theorems are not cumbersome to prove. The equivalence transformation system, on the other hand, provided as axioms all the useful basic equivalences. Secondly, such systems are being used more and more in mechanical verification systems, and the computer science student should be familiar with them. (A natural deduction system is also used in the popular game WFF'N PROOF.) Finally, it is useful to see and compare two totally different formal systems for dealing with propositions.

3.1 Introduction to Deductive Proofs

Consider the problem of proving that a conclusion follows from certain premises. For example, we might want to prove that $p \wedge (r \vee q)$ follows from $p \wedge q$ —i.e. $p \wedge (r \vee q)$ is true in every state in which $p \wedge q$ is. This problem can be written in the following form:

(3.1.1) premise: $p \wedge q$
conclusion: $p \wedge (r \vee q)$

In English, we might argue as follows.

(3.1.2) *Proof of (3.1.1)*: Since $p \wedge q$ is true (in state s), so is p, and so is q. One property of **or** is that, for any r, $r \vee q$ is true if q is, so $r \vee q$ is true. Finally, since p and $r \vee q$ are both true, the properties of **and** allow us to conclude that $p \wedge (r \vee q)$ is true in s also.

In order to get at the essence of such proofs, in order to determine just what is involved in such arguments, we are going to strip away the verbiage from the proof and present simply the bare details. Admittedly, the proofs will look (at first) complicated and detailed. But once we have worked with the proof method for a while, we will be able to return to informal proofs in English with much better facility. We will also be able to give some guidelines for developing proofs (section 3.5).

The bare details of proof (3.1.2) are, in order: a statement of the theorem, the sequence of propositions initially assumed to be true, and the sequence of propositions that are true based on previous propositions and various rules of inference.

These bare details are presented in (3.1.3). The first line states the theorem to be proved: "**From** $p \wedge q$ **infer** $p \wedge (r \vee q)$". The second line gives the premise (if there were more premises, they would be given on successive lines). Each of the succeeding lines gives a proposition that one can infer, based on the truth of the propositions in the previous lines and an inference rule. The last line contains the conclusion.

(3.1.3)

From $p \wedge q$ **infer** $p \wedge (r \vee q)$		
1	$p \wedge q$	premise
2	p	property of **and**, 1
3	q	property of **and**, 1
4	$r \vee q$	property of **or**, 3
5	$p \wedge (r \vee q)$	property of **and**, 2, 4

To the right of each proposition appears an explanation of how the proposition's "truth" is derived. For example, line 4 of the proof indicates that $r \vee q$ is true because of a property of **or** —that $r \vee q$ is true if q is— and because q appears on the preceding line 3. Note that parentheses are introduced freely in order to maintain priority of operators. We shall continue to do this without formal description.

In this formal system, a theorem to be proved has the form

From $e_1, \cdots, e_n$ **infer** e.

In terms of evaluation of propositions, such a theorem is interpreted as: if $e_1, \ldots, e_n$ are true in a state, then e is true in that state also. If n is 0, meaning that there are no premises, then it can be interpreted as: e is true in all states, i.e. e is a tautology. In this case we write it as

Infer e.

Finally, a proposition on a line of a proof can be interpreted to mean that it is true in any state in which the propositions on previous lines are true.

As mentioned earlier, our natural deduction system has no axioms. The properties of operators used above are captured in the inference rules, which we begin to introduce and explain in the next section. (Inference rules were first introduced in section 2.3; review that material if necessary.) The inference rules for the natural deduction system are collected in Figure 3.3.1 at the end of section 3.3.

3.2 Inference Rules

There are ten inference rules in the natural deduction system. Ten is a rather large number, and we can work with that many only if they are organized so that they are easy to remember. In this system, there are two inference rules for each of the five operators **not**, **and**, **or**, **imp** and **equals**. One of the rules allows the introduction of the operator in a new proposition; the other allows its elimination. Hence there are five rules of introduction and five rules of elimination. The rules for introducing and eliminating **and** are called $\wedge$-I and $\wedge$-E, respectively, and similarly for the other operators.

Inference rules $\wedge$-I, $\wedge$-E and $\vee$-I

Let us begin by giving three rules: $\wedge$-I, $\wedge$-E and $\vee$-I.

(3.2.1) $\wedge$-I: $\dfrac{E_1, \cdots, E_n}{E_1 \wedge \cdots \wedge E_n}$

(3.2.2) $\wedge$-E: $\dfrac{E_1 \wedge \cdots \wedge E_n}{E_i}$

(3.2.3) $\vee$-I: $\dfrac{E_i}{E_1 \vee \cdots \vee E_n}$

Rule $\wedge$-I indicates that if E_1 and E_2 occur on previous lines of a proof (i.e. are assumed to be true or have been proved to be true), then their conjunction may be written on a line. If we assert "it is raining", and we assert "the sun is shining", then we can conclude "it is raining and the sun is shining". The rule is called "$\wedge$-Introduction", or "$\wedge$-I" for short, because it shows how a conjunction can be introduced.

Rule $\wedge$-E shows how **and** can be eliminated to yield one of its conjuncts. If $E_1 \wedge E_2$ appears on a previous line of a proof (i.e. is assumed to be true or has been proved to be true), then either E_1 or E_2 may be written on the next line. Based on the assumption "it is raining and the sun is shining", we can conclude "it is raining", and we can conclude "the sun is shining".

Remark: There *are* places where it frequently rains while the sun is shining. Ithaca, the home of Cornell University, is one of them. In fact, it sometimes rains when perfectly blue sky seems to be overhead. The weather can also change from a furious blizzard to bright, calm sunshine and then back again, within minutes. When the weather acts so strangely, as it often does, one says that it is *Ithacating*. □

Rule $\vee$-I indicates that if E_1 is on a previous line, then we may write $E_1 \vee E_2$ on a line. If we assert "it is raining", then we can conclude "it is raining or the sun is shining".

Remember, these rules hold for *all* propositions E_1 and E_2. They are really "schemas", and we get an instance of the rule by replacing E_1 and E_2 by particular propositions. For example, since $p \vee q$ and r are propositions, the following is an instance of $\wedge$-I.

$$\frac{p \vee q, \neg r}{(p \vee q) \wedge \neg r}$$

Let us redo proof (3.1.3) in (3.2.4) below and indicate the exact inference rule used at each step. The top line states what is to be proved. The line numbered 1 contains the first (and only) premise (pr 1). Each other line has the following property. Let the line have the form

line # | E "name of rule", line #, ..., line #

Then one can form an instance of the named inference rule by writing the propositions on lines line #, ..., line # above a line and proposition E below. That is, the truth of E is inferred by one inference rule from the truth of previous propositions. For example, from line 4 of the proof we see that $q \,/\, r \vee q$ is an instance of rule $\vee$-I: $(r \vee q)$ is being inferred from q.

(3.2.4) **From** $p \wedge q$ **infer** $p \wedge (r \vee q)$

1	$p \wedge q$	pr 1
2	p	$\wedge$-E, 1
3	q	$\wedge$-E, 1
4	$r \vee q$	$\vee$-I, 3
5	$p \wedge (r \vee q)$	$\wedge$-I, 2, 4

Note how rule $\wedge$-E is used to break a proposition into its constituent parts, while $\wedge$-I and $\vee$-I are used to build new ones. This is typical of the use of introduction and elimination rules.

Proofs (3.2.5) and (3.2.6) below illustrate that **and** is a commutative operation; if $p \wedge q$ is true then so is $q \wedge p$, and vice versa. This is obvious after our previous study of propositions, but it must be proved in this formal system before it can be used. Note that both proofs are necessary; one cannot derive the second as an instance of the first by replacing p and q in the first by q and p, respectively. In this formal system, a proof holds only for the particular propositions involved. It is not a schema, the way an inference rule is.

(3.2.5) **From** $p \wedge q$ **infer** $q \wedge p$

1	$p \wedge q$	pr 1
2	p	$\wedge$-E, 1
3	q	$\wedge$-E, 1
4	$q \wedge p$	$\wedge$-I, 3, 2

To illustrate the relation between the proof system and English, we give an argument in English for lemma (3.2.5): Suppose $p \wedge q$ is true [line 1]. Then so is p, and so is q [lines 2 and 3]. Therefore, by the definition of **and**, $q \wedge p$ is true [line 4].

(3.2.6) **From** $q \wedge p$ **infer** $p \wedge q$

1	$q \wedge p$	pr 1
2	q	$\wedge$-E, 1
3	p	$\wedge$-E, 1
4	$p \wedge q$	$\wedge$-I, 3, 2

Proof (3.2.6) can be abbreviated by omitting lines containing premises and

using "pr i" to refer to the i^{th} premise later on, as shown in (3.2.7). This abbreviation will occur often. But note that this is only an abbreviation, and we will continue to use the phrase "occurs on a previous line" to include the premises, even though the abbreviation is used.

(3.2.7) **From** $q \wedge p$ **infer** $p \wedge q$

1	q	$\wedge$-E, pr 1
2	p	$\wedge$-E, pr 1
3	$p \wedge q$	$\wedge$-I, 2, 1

Inference rule $\vee$*-E*

The inference rule for elimination of **or** is

$$(3.2.8)\quad \vee\text{-E:}\ \frac{E_1 \vee \cdots \vee E_n,\ E_1 \Rightarrow E,\ \cdots,\ E_n \Rightarrow E}{E}$$

Rule $\vee$-E indicates that if a disjunction appears a previous line, and if $E_i \Rightarrow E$ appears on a previous line for each disjunct E_i, then E may be written on a line of the proof. If we assert "it will rain tomorrow or it will snow tomorrow", and if we assert "rain implies no sun", and if we also assert "snow implies no sun", then we can conclude "there will be no sun tomorrow". From

$$(rain \vee snow),\ (rain \Rightarrow no\ \ sun),\ (snow \Rightarrow no\ \ sun)$$

we conclude *no sun*.

Here is a simple example.

From $p \vee (q \wedge r),\ p \Rightarrow s,\ (q \wedge r) \Rightarrow s$ **infer** $s \vee p$

1	$p \vee (q \wedge r)$	pr 1
2	$p \Rightarrow s$	pr 2
3	$(q \wedge r) \Rightarrow s$	pr 3
4	s	$\vee$-E, 1, 2, 3
5	$s \vee p$	$\vee$-I (rule (3.2.3)), 4

Inference rule $\Rightarrow$*-E*

$$(3.2.9)\quad \Rightarrow\text{-E:}\ \frac{E1 \Rightarrow E2,\ E1}{E2}$$

Rule $\Rightarrow$-E is called *modus ponens*. It allows us to write the consequent of an implication on a line of the proof if its antecedent appears on a previous line. If we assert that $x > 0$ implies that y is even, and if we determine that $x > 0$, then we can conclude that y is even.

We show an example of its use in proof (3.3.10). To show the relation between the formal proof and an English one, we give the proof in English: Suppose $p \wedge q$ and $p \Rightarrow r$ are both true. From $p \wedge q$ we conclude that p is true. Because $p \Rightarrow r$, the truth of p implies the truth of r, and r is true. But if r is true, so is r "ored" with anything; hence $r \vee (q \Rightarrow r)$ is true.

(3.2.10) **From** $p \wedge q,\ p \Rightarrow r$ **infer** $r \vee (q \Rightarrow r)$

1	$p \wedge q$	pr 1
2	$p \Rightarrow r$	pr 2
3	p	$\wedge$-E (rule (3.2.2)), 1
4	r	$\Rightarrow$-E, 2, 3
5	$r \vee (q \Rightarrow r)$	$\vee$-I (rule (3.2.3)), 4

To emphasize the use of the abbreviation to refer to premises, we show (3.2.10) in its abbreviated form in (3.2.11).

(3.2.11) **From** $p \wedge q,\ p \Rightarrow r$ **infer** $r \vee (q \Rightarrow r)$

1	p	$\wedge$-E, pr 1
2	r	$\Rightarrow$-E, pr 2, 1
3	$r \vee (q \Rightarrow r)$	$\vee$-I, 2

Inference rules =-I and =-E

(3.2.12) $$=\text{-I:}\ \frac{E1 \Rightarrow E2,\ E2 \Rightarrow E1}{E1 = E2}$$

(3.2.13) $$=\text{-E:}\ \frac{E1 = E2}{E1 \Rightarrow E2,\ E2 \Rightarrow E1}$$

Rules =-I and =-E together define equality in terms of implication. The premises of one rule are the conclusions of the other, and vice versa. This is quite similar to how equality is defined in the system of chapter 2. Rule =-I is used, then, to introduce an equality $e1 = e2$ based on the previous proof of $e1 \Rightarrow e2$ and $e2 \Rightarrow e1$.

Here is an example of the use of these rules.

From $p,\ p = (q \Rightarrow r),\ r \Rightarrow q$ **infer** $r = q$

1	$p \Rightarrow (q \Rightarrow r)$	=-E, pr 2
2	$q \Rightarrow r$	$\Rightarrow$-E, 1, pr 1
3	$r = q$	=-I, pr 3, 2

Exercises for Section 3.2

1. Each of the following theorems can be proven using exactly one basic inference rule (using the abbreviation that premises need not be written on lines; see the text preceding (3.2.7)). Name that inference rule.

(a) **From** $a,\ b$ **infer** $a \wedge b$
(b) **From** $a \wedge b \wedge (q \vee r),\ a$ **infer** $q \vee r$
(c) **From** $\neg a$ **infer** $\neg a \vee a$
(d) **From** $c = d,\ d \vee c$ **infer** $d \Rightarrow c$
(e) **From** $b \Rightarrow c,\ b$ **infer** $b \vee \neg b$
(f) **From** $\neg a,\ \neg b,\ c$ **infer** $\neg a \vee c$
(g) **From** $(a \Rightarrow b) \wedge b,\ a$ **infer** $a \Rightarrow b$
(h) **From** $a \vee b \Rightarrow c,\ c \Rightarrow a \vee b$ **infer** $a \vee b = c$
(i) **From** $a \wedge b,\ q \vee r$ **infer** $(a \wedge b) \wedge (q \vee r)$
(j) **From** $p \Rightarrow (q \Rightarrow r),\ p,\ q \vee r$ **infer** $q \Rightarrow r$
(k) **From** $c \Rightarrow d,\ d \Rightarrow e,\ d \Rightarrow c$ **infer** $c = d$
(l) **From** $a \vee b,\ a \vee c,\ (a \vee b) \Rightarrow c$ **infer** c
(m) **From** $a \Rightarrow (d \vee c),\ (d \vee c) \Rightarrow a$ **infer** $a = (d \vee c)$
(n) **From** $(a \vee b) \Rightarrow c,\ (a \vee d) \Rightarrow c,\ (a \vee b) \vee (a \vee d)$ **infer** c
(o) **From** $a \Rightarrow (b \vee c),\ b \Rightarrow (b \vee c),\ a \vee b$ **infer** $b \vee c$

2. Here is one proof that p follows from p. Write another proof that uses only one reference to the premise.

From p **infer** p

1	p	pr 1
2	p	pr 1

3. Prove the following theorems using the inference rules.

(a) **From** $p \wedge q,\ p \Rightarrow r$ **infer** r
(b) **From** $p = q,\ q$ **infer** p
(c) **From** $p,\ q \Rightarrow r,\ p \Rightarrow r$ **infer** $p \wedge r$
(d) **From** $b \wedge \neg c$ **infer** $\neg c$
(e) **From** b **infer** $b \vee \neg c$
(f) **From** $b \Rightarrow c \wedge d,\ b$ **infer** d
(g) **From** $p \wedge q,\ p \Rightarrow r$ **infer** r
(h) **From** $p,\ q \wedge (p \Rightarrow s)$ **infer** $q \wedge s$
(i) **From** $p = q$ **infer** $q = p$
(j) **From** $b \Rightarrow (c \wedge d),\ b$ **infer** d

4. For each of your proofs of exercise 3, give an English version. (The English versions need not mimic the formal proofs exactly.)

3.3 Proofs and Subproofs

Inference rule $\Rightarrow$-I

A theorem of the form "**From** $e_1 \cdots, e_n$ **infer** e" is interpreted as: if $e_1, ..., e_n$ are true in a state, then so is e. If $e_1, ..., e_n$ appear on lines of a proof, which is interpreted to mean that they are assumed or proven true, then we should be able to write e on a line also. Rule $\Rightarrow$-I, (3.3.1), gives us permission to do so. Its premise need not appear on a previous line of the proof; it can appear elsewhere as a separate proof, which we refer to in substantiating the use of the rule. Unique names should be given to proofs to avoid ambiguous references.

(3.3.1) $\Rightarrow$-I: $$\frac{\textbf{From } E_1, \cdots, E_n \textbf{ infer } E}{(E_1 \wedge \cdots \wedge E_n) \Rightarrow E}$$

Proof (3.3.2) uses $\Rightarrow$-I twice in order to prove that $p \wedge q$ and $q \wedge p$ are equivalent, using lemmas proved in the previous section.

(3.3.2) **Infer** $(p \wedge q) = (q \wedge p)$

1	$(p \wedge q) \Rightarrow (q \wedge p)$	$\Rightarrow$-I, (3.2.5)
2	$(q \wedge p) \Rightarrow (p \wedge q)$	$\Rightarrow$-I, (3.2.6)
3	$(p \wedge q) = (q \wedge p)$	$=$-I, 1, 2

Rule $\Rightarrow$-I allows us to conclude $p \Rightarrow q$ if we have a proof of q given premise p. On the other hand, if we take $p \Rightarrow q$ as a premise, then rule $\Rightarrow$-E allows us to conclude that q holds when p is given. We see that the following relationship holds:

> **Deduction Theorem.** "**Infer** $p \Rightarrow q$" is a theorem of the natural deduction system, which can be interpreted to mean that $p \Rightarrow q$ is a tautology, *iff* "**From** p **infer** q" is a theorem. □

Another example of the use of $\Rightarrow$-I shows that p implies itself:

(3.3.3) **Infer** $p \Rightarrow p$

1	$p \Rightarrow p$	$\Rightarrow$-I, exercise 2 of section 3.2

Subproofs

A proof can be included within a proof, much the way a procedure can be included within a program. This allows the premise of $\Rightarrow$-I to appear as a line of a proof. To illustrate this, (3.3.2) is rewrit'en i. (3.3.4) to include proof (3.2.5) as a subproof. The subproof happens to be on line 1 here, but it could be on any line. If the subtheorem appears on line j (say) of the main proof, then its proof appears indented underneath, with its lines numbered $j.1$, $j.2$, etc. We could have replaced the reference to (3.2.6) by a subproof in a similar manner.

(3.3.4) **Infer** $(p \wedge q) = (q \wedge p)$

1	**From** $p \wedge q$ **infer** $q \wedge p$	
1.1	p	$\wedge$-E, pr 1
1.2	q	$\wedge$-E, pr 1
1.3	$q \wedge p$	$\wedge$-I, 1.2, 1.1
2	$(p \wedge q) \Rightarrow (q \wedge p)$	$\Rightarrow$-I, 1
3	$(q \wedge p) \Rightarrow (p \wedge q)$	$\Rightarrow$-I, (3.2.6)
4	$(p \wedge q) = (q \wedge p)$	$=$-I, 2, 3

Another example of a proof with a subproof is given in (3.3.5). Again, it may be instructive to compare the proof to an English version:

> Suppose $(q \vee s) \Rightarrow (p \wedge q)$. To prove equivalence, we must show also that $(p \wedge q) \Rightarrow (q \vee s)$. [Note how this uses rule $=$-I, that $a \Rightarrow b$ and $b \Rightarrow a$ means $a = b$. These sentences correspond to lines 1, 3 and 4 of the formal proof.] To prove $(p \wedge q) \Rightarrow (q \vee s)$, argue as follows. Assume $p \wedge q$ is true. Then so is q. By the definition of **or**, so is $q \vee s$. [Note the correspondence to lines 2.1-2.2.] □

(3.3.5) **From** $(q \vee s) \Rightarrow (p \wedge q)$ **infer** $(q \vee s) = (p \wedge q)$

1	$(q \vee s) \Rightarrow (p \wedge q)$	pr 1
2	**From** $p \wedge q$ **infer** $q \vee s$	
2.1	q	$\wedge$-E, pr 1
2.2	$q \vee s$	$\vee$-I, 2.1
3	$(p \wedge q) \Rightarrow (q \vee s)$	$\Rightarrow$-I, 2
4	$(q \vee s) = (p \wedge q)$	$=$-I, 1, 3

As mentioned earlier, the relationship between proofs and sub-proofs in logic is similar to the relationship between procedures and sub-procedures (modules and sub-modules) in programs. A theorem and its proof can be used in two ways: first, use the theorem to prove something else; secondly, study the proof of the theorem. A procedure and its description can be used in two ways: first, understand the description so that calls of the procedure can be written; secondly, study the procedure body to understand how the procedure works. This similarity should make the idea of subproofs easy to understand.

Scope rules

A subproof can contain references not only to previous lines in its proof, but also to previous lines that occur in surrounding proofs. We call these *global* line references. However, "recursion" is not allowed; a line j (say) may not contain a reference to a theorem whose proof is not finished by line j.

The reader skilled in the use of block structure in languages like PL/I, ALGOL 60 and Pascal will have no difficulty in understanding this scope rule, for essentially the same scope mechanism is employed here (except for the restriction against recursion). Let us state the rule more precisely.

(3.3.6) **Scope rule.** Line i of a proof, where i is an integer, may contain references to lines 1, ..., $i-1$. Line $j.i$, where i is an integer, may contain references to lines $j.1$, ..., $j.(i-1)$ and to any lines referenceable from line j (this excludes references to line j itself). □

Example (3.3.7) illustrates the use of this scope rule; line 2.2 refers to line 1, which is outside the proof of line 2.

(3.3.7)

From $p \Rightarrow (q \Rightarrow r)$ **infer** $(p \wedge q) \Rightarrow r$

1	$p \Rightarrow (q \Rightarrow r)$	pr 1
2	**From** $p \wedge q$ **infer** r	
2.1	p	$\wedge$-E, pr 1
2.2	$q \Rightarrow r$	$\Rightarrow$-E, 1, 2.1
2.3	q	$\wedge$-E, pr 1
2.4	r	$\Rightarrow$-E, 2.2, 2.3
3	$(p \wedge q) \Rightarrow r$	$\Rightarrow$-I, 2

Below we illustrate an invalid use of the scope rule.

From p **infer** $p \Rightarrow \neg p$ (Proof INVALID)

1	p	pr 1
2	**From** p **infer** $\neg p$	
2.1	p	pr 1
2.2	$p \Rightarrow \neg p$	$\Rightarrow$-I, 2 (invalid reference to line 2)
2	$p \Rightarrow \neg p$	$\Rightarrow$-I, 2 (valid reference to line 2)

We illustrate another common mistake below; the use of a line that is not in a surrounding proof. Below, on line 6.1 an attempt is made to reference s on line 4.1. Since line 4.1 is not in a *surrounding* proof, this is not allowed.

A subproof using global references is being proved *in a particular context*. Taken out of context, the subproof may not be true because it relies

From $p \vee q, p \Rightarrow s, s \Rightarrow r$ **infer** r (proof INVALID)

1	$p \vee q$	pr 1
2	$p \Rightarrow s$	pr 2
3	$s \Rightarrow r$	pr 3
4	**From** p **infer** r	
4.1	s	$\Rightarrow$-E, 2, pr 1 (valid reference to 2)
4.2	r	$\Rightarrow$-E, 3, 4.1 (valid reference to 3)
5	$p \Rightarrow r$	$\Rightarrow$-I, 4
6	**From** q **infer** r	
6.1	r	$\Rightarrow$-E, 3, 4.1 (invalid reference to 4.1)
7	$q \Rightarrow r$	$\Rightarrow$-I, 6
8	r	$\vee$-E, 1, 5, 7

on assumptions about the context. This again points up the similarity between ALGOL-like procedures and subproofs. Facts assumed outside a subproof can be used within the proof, just as variables declared outside a procedure can be used within a procedure, using the same scope mechanism.

To end this discussion of scope, we give a proof with two levels of subproof. It can be understood most easily as follows. First read lines 1, 2 and 3 (don't read the the proof of the lemma on line 2) and satisfy yourself that *if* the proof of the lemma on line 2 is correct, then the whole proof is correct. Next, study the proof of the lemma on line 2 (only lines 2.1, 2.2 and 2.3). Finally, study the proof of the lemma on line 2.2, which refers to a line two levels out in the proof.

(3.3.8)

From $(p \wedge q) \Rightarrow r$ **infer** $p \Rightarrow (q \Rightarrow r)$

1	$(p \wedge q) \Rightarrow r$	pr 1
2	**From** p **infer** $q \Rightarrow r$	
2.1	p	pr 1
2.2	**From** q **infer** r	
2.2.1	$p \wedge q$	$\wedge$-I, 2.1, pr 1
2.2.2	r	$\Rightarrow$-E, 1, 2.2.1
2.3	$q \Rightarrow r$	$\Rightarrow$-I, 2.2
3	$p \Rightarrow (q \Rightarrow r)$	$\Rightarrow$-I, 2

Proof by contradiction

A proof by contradiction typically proceeds as follows. One makes an assumption. From this assumption one proceeds to prove a contradiction, say, by showing that something is both true and false. Since such a contradiction cannot possibly happen, and since the proof from assumption to contradiction is valid, the assumption must be false.

Proof by contradiction is embodied in the proof rules $\neg$-I and $\neg$-E:

$$(3.3.9) \quad \neg\text{-I: } \frac{\textbf{From } E \textbf{ infer } E1 \wedge \neg E1}{\neg E}$$

$$(3.3.10) \quad \neg\text{-E: } \frac{\textbf{From } \neg E \textbf{ infer } E1 \wedge \neg E1}{E}$$

Rule $\neg$-I indicates that if "**From** E **infer** $E1 \wedge \neg E1$" has been proved for some proposition $E1$, then one can write $\neg E$ on a line of the proof.

Rule $\neg$-I similarly allows us to conclude that E holds if a proof of "**From** $\neg E$ **infer** $E1 \wedge \neg E1$" exists, for some proposition $E1$.

We show in (3.3.11) an example of the use of rule $\neg$-I, that from p we can conclude $\neg\neg p$.

		From p **infer** $\neg\neg p$	
	1	p	pr 1
(3.3.11)	2	**From** $\neg p$ **infer** $p \wedge \neg p$	
	2.1	$p \wedge \neg p$	$\wedge$-I, 1, pr 1
	3	$\neg\neg p$	$\neg$-I, 2

Rule $\neg$-I is used to prove that $\neg\neg p$ follows from p; similarly, rule $\neg$-E is used in (3.3.12) to prove that p follows from $\neg\neg p$.

		From $\neg\neg p$ **infer** p	
	1	$\neg\neg p$	pr 1
(3.3.12)	2	**From** $\neg p$ **infer** $\neg p \wedge \neg\neg p$	
	2.1	$\neg p \wedge \neg\neg p$	$\wedge$-I, pr 1, 1
	3	p	$\neg$-E, 2

Theorems (3.3.11) and (3.3.12) look quite similar, and yet both proofs are needed; one cannot simply get one from the other more easily than they are proven here. More importantly, both of the *rules* $\neg$-I and $\neg$-E are needed; if one is omitted from the proof system, we will be unable to deduce some propositions that are tautologies in the sense described in section 1.5. This may seem strange, since the rules look so similar.

Let us give two more proofs. The first one indicates that from p and $\neg p$ one can prove *any* proposition q, even one that is equivalent to false. This is because both p and $\neg p$ cannot both be true at the same time, and hence the premises form an absurdity.

		From $p, \neg p$ **infer** q	
	1	p	pr 1
	2	$\neg p$	pr 2
(3.3.13)	3	**From** $\neg q$ **infer** $p \wedge \neg p$	
	3.1	$p \wedge \neg p$	$\wedge$-I, 1, 2
	4	q	$\neg$-E, 3

		From $p \wedge q$ **infer** $\neg(p \Rightarrow \neg q)$	
	1	$p \wedge q$	pr 1
	2	**From** $p \Rightarrow \neg q$ **infer** $q \wedge \neg q$	
(3.3.14)	2.1	p	$\wedge$-E, 1
	2.2	q	$\wedge$-E, 1
	2.3	$\neg q$	$\Rightarrow$-E, pr 1, 2.1
	2.4	$q \wedge \neg q$	$\wedge$-I, 2.2, 2.3
	3	$\neg(p \Rightarrow \neg q)$	$\neg$-I, 2

For comparison, we give an English version of proof (3.3.14). Let $p \wedge q$ be true. Then both p and q are true. Assume that $p \Rightarrow \neg q$ is true. Because p is true this implication allows us to conclude that $\neg q$ is true, but this is absurd because q is true. Hence the assumption that $p \Rightarrow \neg q$ is true is wrong, and $\neg(p \Rightarrow \neg q)$ holds.

Summary

The reader may have noticed a difference between the natural deduction system and the previous systems of evaluation and equivalence transformation: the natural deduction system does not allow the use of constants T and F! The connection between the systems can be stated as follows. If "**Infer** e" is a theorem of the natural deduction system, then e is a tautology and $e = T$ is an equivalence. On the other hand, if $e = T$ is a tautology and e does not contain T and F, then "**Infer** e" is a theorem of the natural deduction system. The omission of T and F is no problem because, by the rule of Substitution, in any proposition T can be replaced by a tautology (e.g. $b \vee \neg b$) and F by the complement of a tautology (e.g. $b \wedge \neg b$) to yield an equivalent proposition.

We summarize what a proof is as follows. A proof of a theorem "**From** $e_1, \cdots, e_n$ **infer** e" or of a theorem "**Infer** e" consists of a sequence of lines. The first line contains the theorem. If the first line is unnumbered, the rest are indented and numbered 1, 2, etc. If the first line has the number i, the rest are indented and numbered $i.1$, $i.2$, etc. The last line must contain proposition e. Each line i must have one of the following four forms:

Form 1: $(i)\ e_j$ pr j
where $1 \leq j \leq n$. The line contains premise j.

Form 2: $(i)\ p$ *Name*, $ref_1, \ldots, ref_q$
Each ref_k either (1) is a line number (which is valid according to scope rule (3.3.6)), or (2) has the form "pr j", in which case it refers to premise e_j of the theorem, or (3) is the name of a previously proven theorem. Let r_k denote the proposition or theorem referred to by ref_k. Then the following must be an instance of inference rule *Name*:

$$\frac{r_1, \cdots, r_q}{p}$$

Form 3: $(i)\ p$ *Theorem name*, $ref_1, \ldots, ref_q$
Theorem name is the name of a previously proved theorem; ref_k is as in Form 2. Let r_k denote the proposition referred to be ref_k. Then "**From** $r_1, \cdots, r_q$ **infer** p" must be the named theorem.

Form 4: (i) [Proof of another theorem]
That is, the line contains a complete subproof, whose format follows these rules.

Figure 3.3.1 contains a list of the inference rules.

Historical Notes

The style of the logical system defined in this chapter was conceived principally to capture our "natural" patterns of reasoning. Gerhard Gentzen, a German mathematician who died in an Allied prisoner of war camp just after World War II, developed such a system for mathematical arguments in his 1935 paper *Untersuchungen ueber das logische Schliessen* [20], which is included in [43].

Several textbooks on logic are based on natural deduction, for example W.V.O. Quine's book *Methods of Logic* [41].

The particular block-structured system given here was developed using two sources: *WFF'N PROOF: The Game of Modern Logic*, by Layman E. Allen [1] and the monograph *A Programming Logic*, by Robert Constable and Michael O'Donnell [7]. The former introduces the deduction system through a series of games; it uses prefix notation, partly to avoid problems with parentheses, which we have sidestepped through informality. *A Programming Logic* describes a mechanical program verifier for

PL/CS (a subset of PL/C, which is a subset of PL/I), developed at Cornell University. Its inference rules were developed with ease of presentation and mechanical verification in mind. Actually, the verifier can be used to verify proofs of programs, and includes not only the propositional calculus but also a predicate calculus, including a theory of integers and a theory of strings.

$$\wedge\text{-I: } \frac{E_1, \ldots, E_n}{E_1 \wedge \ldots \wedge E_n} \qquad \wedge\text{-E: } \frac{E_1 \wedge \ldots \wedge E_n}{E_i}$$

$$\vee\text{-I: } \frac{E_i}{E_1 \vee \ldots \vee E_n} \qquad \vee\text{-E: } \frac{E_1 \vee \ldots \vee E_n,\ E_1 \Rightarrow E,\ \ldots,\ E_n \Rightarrow E}{E}$$

$$\neg\text{-I: } \frac{\textbf{From } E \textbf{ infer } E1 \wedge \neg E1}{\neg E} \qquad \neg\text{-E: } \frac{\textbf{From } \neg E \textbf{ infer } E1 \wedge \neg E1}{E}$$

$$=\text{-I: } \frac{E1 \Rightarrow E2,\ E2 \Rightarrow E1}{E1 = E2} \qquad =\text{-E: } \frac{E1 = E2}{E1 \Rightarrow E2,\ E2 \Rightarrow E1}$$

$$\Rightarrow\text{-I: } \frac{\textbf{From } E_1, \ldots, E_n \textbf{ infer } E}{(E_1 \wedge \ldots \wedge E_n) \Rightarrow E} \qquad \Rightarrow\text{-E: } \frac{E1 \Rightarrow E2,\ E1}{E2}$$

Figure 3.3.1 The Set of Basic Inference Rules

Exercises for Section 3.3

1. Use lemma (3.2.11) and inference rule $\Rightarrow$-I to give a 1-line proof that $(p \wedge q \wedge (p \Rightarrow r)) \Rightarrow (r \vee (q \Rightarrow r))$.

2. Prove that $(p \wedge q) \Rightarrow (p \vee q)$, using rule $\Rightarrow$-I.

3. Prove that $q \Rightarrow (q \wedge q)$. Prove that $(q \wedge q) \Rightarrow q$. Use the first two results to prove that $q = (q \wedge q)$. Then rewrite the last proof so that it does not refer to outside proofs.

4. Prove that $p = (p \vee p)$.

5. Prove that $p \Rightarrow ((r \vee s) \Rightarrow p)$.

6. Prove that $q \Rightarrow (r \Rightarrow (q \wedge r))$.

7. Prove that from $p \Rightarrow (r \Rightarrow s)$ follows $r \Rightarrow (p \Rightarrow s)$.

8. What is wrong with the following proof?

Infer $a \Rightarrow b$ (Proof INVALID)

1		a	pr 1
2		**From** $\neg b$ **infer** $b \wedge \neg b$	
	2.1	$\neg b$	pr 1
	2.2	$\neg b \Rightarrow b \wedge \neg b$	$\Rightarrow$-I, 2
	2.3	$b \wedge \neg b$	$\Rightarrow$-E, 2.2, 2.1
2		b	$\neg$-E, 2

9. Prove that from $\neg p$ and $(\neg p \Rightarrow q) \vee (p \wedge (r \Rightarrow q))$ follows $r \Rightarrow q$.

10. Prove that $q \Rightarrow (p \wedge r)$ follows from $q \Rightarrow p$ and $q \Rightarrow r$.

11. Prove that from $\neg q$ follows $q \Rightarrow p$.

12. Prove that from $\neg q$ follows $q \Rightarrow \neg p$.

13. Prove that from $\neg q$ follows $q \Rightarrow (p \wedge \neg p)$.

14. Prove that from $p \vee q$, $\neg q$ follows p.

15. Prove $p \wedge (p \Rightarrow q) \Rightarrow q$.

16. Prove $((p \Rightarrow q) \wedge (q \Rightarrow r)) \Rightarrow (p \Rightarrow r)$.

17. Prove $(p \Rightarrow q) \Rightarrow ((p \wedge \neg q) \Rightarrow q)$.

18. Prove $((p \wedge \neg q) \Rightarrow q) \Rightarrow (p \Rightarrow q)$. [This, together with exercise 17, allows us to prove $(p \Rightarrow q) = ((p \wedge \neg q) \Rightarrow q)$.]

19. Prove $(p \Rightarrow q) \Rightarrow ((p \wedge \neg q) \Rightarrow \neg p)$.

20. Prove $((p \wedge \neg q) \Rightarrow \neg p) \Rightarrow (p \Rightarrow q)$. [This, together with exercise 19, allows us to prove $(p \Rightarrow q) = ((p \wedge \neg q) \Rightarrow \neg p)$.]

21. Prove that $(p = q) \Rightarrow (\neg p = \neg q)$.

22. Prove that $(\neg p = \neg q) \Rightarrow (p = q)$. [This, together with exercise 21, allows us to prove $(p = q) = (\neg p = \neg q)$.]

23. Prove $\neg(p = q) \Rightarrow (\neg p = q)$

24. Prove $(\neg p = q) \Rightarrow \neg(p = q)$. [This, together with exercise 21, allows us to prove the law of Inequality, $\neg(p = q) = (\neg p = q)$.]

25. Prove $(p = q) \Rightarrow (q = p)$.

26. Use a rule of Contradiction to prove **From** p **infer** p.

27. For each of the proofs of exercise 1-7, 9-25, give a version in English. (It need not follow the formal proof exactly.)

3.4 Adding Flexibility to the Natural Deduction System

We first introduce some flexibility by showing how theorems can be viewed as schemas —i.e. how identifiers in a theorem can be viewed as standing for any arbitrary proposition. Next, we introduce a rule of substitution of equals for equals, incorporating into the natural deduction system the method of proving equivalences of chapter 2. We prove a number of theorems, including the laws of equivalence of chapter 2.

Using theorems as schemas

The inference rules given in Figure 3.3.1 hold for any propositions E, E_1, ..., E_n. They are really "schemas", and one gets a particular inference rule by substituting particular propositions for the "placeholders" E, E_1, ..., E_n. On the other hand, theorems of the form "**From** *premises* **infer** *conclusion*" are proved only for particular propositions. For example, proof (3.3.2) used the following two theorems (3.2.5) and (3.2.6):

From $p \wedge q$ **infer** $q \wedge p$

From $q \wedge p$ **infer** $p \wedge q$

Even though it looks like the second should follow directly from the first, in the formal system both must be proved.

But we can prove something *about* the formal system: systematic substitution of propositions for identifiers in a theorem and its proof yields another theorem and proof. So we can consider any theorem to be a schema also. For example, from proof (3.2.5) of "**From** $p \wedge q$ **infer** $q \wedge p$" we can generate a proof of "**From** $(a \vee b) \wedge c$ **infer** $c \wedge (a \vee b)$" simply by substituting $a \vee b$ for p and c for q everywhere in proof (3.2.5):

From $(a \vee b) \wedge c$ **infer** $c \wedge (a \vee b)$

1	$(a \vee b) \wedge c$	pr 1
2	$a \vee b$	$\wedge$-E, 1
3	c	$\wedge$-E, 1
4	$c \wedge (a \vee b)$	$\wedge$-I, 3, 2

Let us state more precisely this idea of textual substitution in theorem and proof.

(3.4.1) **Theorem.** Write a theorem as a function of one of its identifiers, p: "**From** $E_1(p)$, ..., $E_n(p)$ **infer** $E(p)$". Let G be any proposition. Then "**From** $E_1(G)$, ..., $E_n(G)$ **infer** $E(G)$" can also be proved.

Informal proof. Without loss of generality, assume the proof of the theorem contains no references to other theorems outside the proof. (If it does, first change the proof to include them as subproofs, as was done in generating proof (3.3.4) from proof (3.3.2), repeating the process until no references to outside theorems exist.) Then we can obtain a proof of the new theorem simply by substituting G for p everywhere in the proof of the original theorem. □

Theorems like (3.4.1) are often called *meta*-theorems, because they are not theorems in the proof system, like "**From** ... **infer** ...", but are proofs *about* the proof system. The use of meta-theorems takes us outside the formal system just a bit, but it is worthwhile to relax formality in this way.

We can put meta-theorem (3.4.1) in the form of a *derived rule of inference* as follows:

$$(3.4.2)\quad \frac{\textbf{From } E_1(p), \cdots, E_n(p) \textbf{ infer } E(p)}{\textbf{From } E_1(G), \cdots, E_n(G) \textbf{ infer } E(G)} \quad (p \text{ an identifier})$$

We use this derived rule of inference to rewrite theorem (3.3.2) using only theorem (3.2.5) (and not (3.2.6)). Note how line 2 refers to theorem (3.2.5) and indicates what propositions are being replaced. We often leave out this indication if it is obvious enough.

Infer $(p \wedge q) = (q \wedge p)$

1	$(p \wedge q) \Rightarrow (q \wedge p)$	(3.2.5)
2	$(q \wedge p) \Rightarrow (p \wedge q)$	(3.2.5) (with p for q, q for p)
3	$(p \wedge q) = (q \wedge p)$	=-I, 1, 2

Earlier, we discussed the relation between procedures of a program and subproofs of a proof. We can now extend this relation to procedures with parameters and subproofs with parameters. Consider rule (3.4.2). The proof of the premise corresponds to the definition of a procedure with a parameter p. The use of the conclusion in another proof corresponds to a call of the procedure with an argument G.

The Rule of Substitution of equals for equals

The rule of Substitution, introduced in section 2.2, will be used in this section in the following form.

(3.4.3) **Theorem.** Let proposition E be thought of as a function of one of its identifiers, p, so that we write it as $E(p)$. Then if $e1 = e2$ and $E(e1)$ appear on previous lines, then we may write $E(e2)$ on a line. □

For example, given that $c \Rightarrow a \vee b$ is true, to show that $c \Rightarrow b \vee a$ is true we take $E(p)$ to be $c \Rightarrow p$, $e1 = e2$ to be $a \vee b = b \vee a$ (the law of Commutativity, which will be proved later) and apply the theorem.

The rule of Substitution was an inference rule in the equivalence system of chapter 2. However, it is a meta-theorem of the natural deduction system and must be proved. Its proof, which would be performed by induction on the structure of proposition $E(p)$, is left to the interested reader in exercise 10, so let us suppose it has been done. We put the rule of Substitution in the form of a derived inference rule:

$$(3.4.4)\quad \text{subs: } \frac{e1 = e2,\ E(e1)}{E(e2)} \quad (E(p) \text{ is a function of } p)$$

To show the use of (3.4.4), we give a schematic proof to show that the rule of substitution as given in section 2.2 holds here also.

(3.4.5)

From $e1 = e2$ **infer** $E(e1) = E(e2)$

1	$e1 = e2$	pr 1
2	**From** $E(e1)$ **infer** $E(e2)$	
2.1	$E(e2)$	subs, pr 1, 1
3	$E(e1) \Rightarrow E(e2)$	⇒-I, 2
4	**From** $E(e2)$ **infer** $E(e1)$	
4.1	$e2 = e1$	=-I, (3.3.3) $(p \Rightarrow p)$
4.2	$E(e1)$	subs, 4.1, pr 1
5	$E(e2) \Rightarrow E(e1)$	⇒-I, 4
6	$E(e1) = E(e2)$	=-I, 3, 5

With this derived rule of inference, we have the flexibility of both the equivalence and the natural deduction systems. But we must make sure that the laws of section 2.1 actually hold! We do this next.

Some basic theorems

A number of theorems are used often, including the laws of section 2.1. We want to state them here and prove some of them; the rest of the proofs are left as exercises. The first to be proved is quite useful. It states that if at least one of two propositions is true, and if the first is false, then the second is true.

```
         From p ∨ q, ¬p  infer  q
         1   | ¬p                         pr 2
         2   | From p  infer q
             |  2.1 | p                   pr 1
(3.4.6)      |  2.2 | From ¬q  infer p ∧ ¬p
             |      |  2.2.1 | p ∧ ¬p     ∧-I, 2.1, 1
             |  2.3 | q                   ¬-E, 2.2
         3   | p ⇒ q                      ⇒-I, 2
         4   | q                          ∨-E, pr 1, 3, (3.3.3)
```

We now turn to the laws of section 2.1. Some of their proofs are given here; the others are left as exercises to the reader.

1. *Commutative laws.* $(p \wedge q) = (q \wedge p)$ was proven in theorem (3.3.4); the other two commutative laws are left to the reader to prove.

2. *Associative laws.* These we don't need to prove since the inference rules for **and** and **or** were written using any number of operands and no parentheses.

3. *Distributive laws.* Here is a proof of the first; the second is left to the reader. The proof is broken into three parts. The first part proves an implication $\Rightarrow$ and the second part proves it in the other direction, so that the third can prove the equivalence. The second part uses a case analysis (rule ∨-E) on $b \vee \neg b$ —the law of the Excluded Middle— which is not proved until later. The use of $b \vee \neg b$ in this fashion occurs often

```
         From b ∨ (c ∧ d) infer (b ∨ c) ∧ (b ∨ d)
         1  | From b  infer (b ∨ c) ∧ (b ∨ d)
            |  1.1 | b ∨ c                     ∨-I, pr 1
            |  1.2 | b ∨ d                     ∨-I, pr 1
(3.4.7)     |  1.3 | (b ∨ c) ∧ (b ∨ d)         ∧-I, 1.1, 1.2
         2  | b ⇒ (b ∨ c) ∧ (b ∨ d)            ⇒-I, 1
         3  | From c ∧ d  infer (b ∨ c) ∧ (b ∨ d)
            |  3.1 | c                         ∧-E, pr 1
            |  3.2 | d                         ∧-E, pr 1
            |  3.3 | b ∨ c                     ∨-I, 3.1
            |  3.4 | b ∨ d                     ∨-I, 3.2
            |  3.5 | (b ∨ c) ∧ (b ∨ d)         ∧-I, 3.3, 3.4
         4  | (c ∧ d) ⇒ (b ∨ c) ∧ (b ∨ d)      ⇒-I, 3
         5  | (b ∨ c) ∧ (b ∨ d)                ∨-E, pr 1, 2, 4
```

```
           From (b ∨ c) ∧ (b ∨ d) infer b ∨ (c ∧ d)
           1  | b ∨ c                       ∧-E, pr 1
           2  | b ∨ d                       ∧-E, pr 1
           3  | b ∨ ¬b                      (3.4.14)
           4  | From b  infer b ∨ (c ∧ d)
              |  4.1 | b ∨ (c ∧ d)          ∨-I, pr 1
           5  | b ⇒ b ∨ (c ∧ d)             ⇒-I, 4
(3.4.8)    6  | From ¬b  infer b ∨ (c ∧ d)
              |  6.1 | c                    (3.4.6), 1, pr 1
              |  6.2 | d                    (3.4.6), 2, pr 1
              |  6.3 | c ∧ d                ∧-I, 6.1, 6.2
              |  6.4 | b ∨ (c ∧ d)          ∨-I, 6.3
           7  | ¬b ⇒ b ∨ (c ∧ d)            ⇒-I, 6
           8  | b ∨ (c ∧ d)                 ∨-E, 3, 5, 7
```

```
(3.4.9)  Infer b ∨ (c ∧ d) = (b ∨ c) ∧ (b ∨ d)
         1 | b ∨ (c ∧ d) ⇒ (b ∨ c) ∧ (b ∨ d)     ⇒-I, (3.4.7)
         2 | (b ∨ c) ∧ (b ∨ d) ⇒ b ∨ (c ∧ d)     ⇒-I, (3.4.8)
         3 | b ∨ (c ∧ d) = (b ∨ c) ∧ (b ∨ d)     =-I, 1, 2
```

4. *De Morgans's laws.* We prove only the first one here.

```
           From ¬(b ∧ c) infer ¬b ∨ ¬c
           1  | ¬(b ∧ c)                                        pr 1
           2  | From ¬(¬b ∨ ¬c) infer (b ∧ c) ∧ ¬(b ∧ c)
              |  2.1 | ¬(¬b ∨ ¬c)                               pr 1
              |  2.2 | From ¬b  infer (¬b ∨ ¬c) ∧ ¬(¬b ∨ ¬c)
(3.4.10)      |      |  2.2.1 | ¬b ∨ ¬c                         ∨-I, pr 1
              |      |  2.2.2 | (¬b ∨ ¬c) ∧ ¬(¬b ∨ ¬c)          ∧-I, 2.2.1, 2.1
              |  2.3 | b                                        ¬-E, 2.2
              |  2.4 | From ¬c  infer (¬b ∨ ¬c) ∧ ¬(¬b ∨ ¬c)
              |      |  2.4.1 | ¬b ∨ ¬c                         ∨-I, pr 1
              |      |  2.4.2 | (¬b ∨ ¬c) ∧ ¬(¬b ∨ ¬c)          ∧-I, 2.4.1, 2.1
              |  2.5 | c                                        ¬-E, 2.4
              |  2.6 | b ∧ c                                    ∧-I, 2.3, 2.5
              |  2.7 | (b ∧ c) ∧ ¬(b ∧ c)                       ∧-I, 2.6, 1
           3  | ¬b ∨ ¬c                                         ¬-E, 2
```

```
          From ¬b ∨ ¬c infer ¬(b ∧ c)
          1 | From ¬b infer ¬(b ∧ c)
            | 1.1 | ¬b                        pr 1
            | 1.2 | From b ∧ c infer b ∧ ¬b
            |     | 1.2.1 | b                 ∧-E, pr 1
(3.4.11)    |     | 1.2.2 | b ∧ ¬b            ∧-I, 1.2.1, 1.1
            | 1.3 | ¬(b ∧ c)                  ¬-I, 1.2
          2 | ¬b ⇒ ¬(b ∧ c)                   ⇒-I, 1
          3 | From ¬c infer ¬(b ∧ c)
            | 3.1 | ¬c                        pr 1
            | 3.2 | From b ∧ c infer c ∧ ¬c
            |     | 3.2.1 | c                 ∧-E, pr 1
            |     | 3.2.2 | c ∧ ¬c            ∧-I, 3.2.1, 3.1
            | 3.3 | ¬(b ∧ c)                  ¬-I, 3.2
          4 | ¬c ⇒ ¬(b ∧ c)                   ⇒-I, 3
          5 | ¬(b ∧ c)                        ∨-E, pr 1, 2, 4
```

```
(3.4.12) Infer ¬(b ∧ c) = ¬b ∨ ¬c
          1 | ¬(b ∧ c) ⇒ ¬b ∨ ¬c      ⇒-I, (3.4.10)
          2 | ¬b ∨ ¬c ⇒ ¬(b ∧ c)      ⇒-I, (3.4.11)
          3 | ¬(b ∧ c) = ¬b ∨ ¬c      =-I, 1, 2
```

5. *Law of Negation.* This one is extremely simple because we have already done the necessary groundwork in previous theorems:

```
(3.4.13) Infer ¬¬b = b
          1 | b ⇒ ¬¬b      ⇒-I, (3.3.11)
          2 | ¬¬b ⇒ b      ⇒-I, (3.3.12)
          3 | ¬¬b = b      =-I, 1, 2
```

6. *Law of the Excluded Middle.* This proof proceeds by assuming the converse and proving a contradiction in a straightforward manner.

```
          Infer b ∨ ¬b
          1 | From ¬(b ∨ ¬b) infer (b ∨ ¬b) ∧ ¬(b ∨ ¬b)
            | 1.1 | ¬(b ∨ ¬b)                                  pr 1
            | 1.2 | From ¬b infer (b ∨ ¬b) ∧ ¬(b ∨ ¬b)
            |     | 1.2.1 | b ∨ ¬b                             ∨-I, pr 1
(3.4.14)    |     | 1.2.2 | (b ∨ ¬b) ∧ ¬(b ∨ ¬b)               ∧-I, 1.2.1, 1.1
            | 1.3 | b                                          ¬-E, 1.2
            | 1.4 | b ∨ ¬b                                     ∨-I, 1.3
            | 1.5 | (b ∨ ¬b) ∧ ¬(b ∨ ¬b)                       ∧-I, 1.4, pr 1
          2 | b ∨ ¬b                                           ¬-E, 1
```

7. *Law of Contradiction.* Left to the reader.

8. *Law of Implication.* Left to the reader.

9. *Law of Equality.* Left to the reader.

10-11. *Laws of* **or**- *and* **and**-*Simplification.* These laws use the constants T and F, which don't appear in the inference system.

Exercises for Section 3.4

1. Use the idea in theorem (3.4.1) to derive from (3.3.7) a proof that $(p \wedge q) \Rightarrow (p \vee q)$ follows from $p \Rightarrow (q \Rightarrow p \vee q)$.

2. Use the idea in theorem (3.4.1) to derive from (3.3.8) a proof that from $(q \wedge r \wedge q) \Rightarrow r$ follows $(q \wedge r) \Rightarrow (q \Rightarrow r)$.

3. Use the idea in theorem (3.4.1) to derive from (3.3.4) a proof that $(a \wedge b \wedge c) = (c \wedge a \wedge b)$.

4. Prove the second and third Commutative laws, $(b \vee c) = (c \vee b)$ and $(b = c) = (c = b)$.

5. Prove the second Distributive law, $b \wedge (c \vee d) = (b \wedge c) \vee (b \wedge d)$.

6. Prove the second of De Morgan's laws, $\neg(b \vee c) = \neg b \wedge \neg c$.

7. Prove the law of Contradiction, $\neg(b \wedge \neg b)$.

8. Prove the law of Implication, $b \vee c = (\neg b \Rightarrow c)$.

9. Prove the law of Equality, $(b = c) = (b \Rightarrow c) \wedge (c \Rightarrow b)$.

10. Prove theorem (3.4.3).

11. Prove the rule of Transitivity: from $a = b$ and $b = c$ follows $a = c$.

12. Prove that from $p \vee q$ and $\neg q$ follows p (see (3.4.6)).

3.5 Developing Natural Deduction System Proofs

The reader has no doubt struggled to prove some theorems in the natural deduction system, and has wondered whether such proofs could be developed in a systematic manner. This section should provide some help.

We will begin to be less formal, stating facts without formal proof and taking larger steps in a proof when doing so does not hamper understanding. This is not only convenient; it is necessary. While the formal methods are indispensable for learning about propositions, one must begin to use the insight they supply instead of the complete formality they require in order to keep from being buried under mounds of detail.

To help the reader take a more active role in the development of the proofs, they will be presented as follows. At each step, a question will be posed, which must be answered in order to invent the next step in the proof. The answer will be separated from the question by white space and an underline, so that the reader can try to answer the question before proceeding. In this way, the reader can actually develop each step of the proof and check it with the one presented.

Some general hints on developing proofs

Suppose a theorem of the form "**From** *e1*, *e2* **infer** *e3*" is to be proved. The proof must have the form

```
From e1, e2 infer e3
1 | e1     pr 1
2 | e2     pr 2
  |
3 | e3     Why?
```

and we need only substantiate line 3 —i.e. give a reason why *e3* can be written on it. We can look to three things for insight. First, we may be able to combine the premises or derive sub-propositions from them in some fashion, if not to produce *e3* at least to get something that looks similar to it.

Secondly, we can investigate *e3* itself. Since an inference rule must be used to substantiate line 3, the form of *e3* should help us decide which inference rule to use. And this leads us to the third piece of information we can use, the inference rules themselves. There are ten inference rules, which yields a lot of possibilities. Fortunately, few of them will apply to any particular proposition *e3*, because *e3* must have the form of the conclusion of the inference rule used to substantiate it. And, with the additional information of the premises, the number of actual possibilities can be reduced even more.

For example, if *e3* has the form *e4*⇒*e5*, the two most likely inference rules to use are =-E and ⇒-I, and if a suitable equivalence does not seem possible to derive from the premises, then =-E can be eliminated from consideration.

Let us suppose we try to substantiate line 3 using rule ⇒-I, because it has the form *e4*⇒*e5*. Then we would expand the proof as follows.

```
From e1, e2 infer e4⇒e5
1 | e1                pr 1
2 | e2                pr 2
3 | From e4 infer e5
  |   3.1 | e4        pr 1
  |       |
  |   3.2 | e5        Why?
4 | e4⇒e5             ⇒-I, 3
```

Thus, we have reduced the problem of proving *e4*⇒*e5* from *e1* and *e2* to the problem of proving *e5* from *e4*, and the new problem promises to be simpler because propositions *e4* and *e5* each contain fewer operations than *e3* did —they are in some sense smaller and simpler.

The above discussion shows basically how to go about developing a proof. At each step, investigate the inference rules to determine which are most likely to be applicable, based mainly on the proposition to be proved and secondly on previous assumptions and already-proved theorems, and attempt to apply one of them in order to reduce the problem to a simpler one.

As the proof expands and more assumptions are made, try to invent and substantiate new propositions (from the already proved ones) that may be helpful for proving the desired result. But remember that, while the premises are certainly useful, proof development is a *goal-oriented* activity, and it is mainly the goal, the proposition that must be substantiated; we should look to the goal and possible inference rules for the most insight.

Successful proof development requires some experience with the inference rules, so the reader should spend some time studying them and deciding when they might be employed. We can give some hints here.

Rules =-I and =-E together define operation **equals**. They are used only to derive an equivalence or to turn one into implications. If equivalence is not a part of the premises or goal, they can be eliminated from consideration.

The other rules of introduction are used to introduce longer propositions from shorter ones. Hence, they are useful when the desired goal, or parts of it, can be built from shorter propositions that occur on previous lines. Note that, except for =-I, the forms of the conclusions of the rules of introduction are all different, so that at most one of these rules can be used to substantiate a proposition.

The rules of elimination are generally used to "break apart" a proposition so that one of its sub-propositions can be derived. All the rules of elimination (except for =-E) have a general proposition as their conclusion. This means that they may possibly be used to substantiate *any* proposition. Whether an elimination rule can be used depends on whether its premises have appeared on previous lines, so to decide whether these rules should be used requires a look at previous lines.

The Development of a proof

Problem. Prove that if $p \Rightarrow q$ is true then so is $(p \land \neg q) \Rightarrow \neg p$. The first step in developing a proof is to draw the outline for the proof and fill in the first line with the theorem, the next lines with the premises and the last line with the goal —i.e the proposition to be inferred. Perform this step.

The problem description yields the following start of a proof:

```
   From p ⇒ q  infer (p ∧ ¬q) ⇒ ¬p
1 | p ⇒ q                 pr 1
  |
2 | (p ∧ ¬q) ⇒ ¬p         Why?
```

At this point, it is wise to study the premises to see whether propositions can be derived from them. Do this.

Little can be derived from $p \Rightarrow q$, except the disjunction $\neg p \lor q$ (using the rule of Substitution). We will keep this proposition in mind. Which rules of inference could be used to substantiate line 2? That is, which rules of inference could have $(p \land \neg q) \Rightarrow \neg p$ as their conclusion?

Possible inference rules are: $\Rightarrow$-I, $\land$-E, $\lor$-E, $\neg$-E, =-E and $\Rightarrow$-E. Which seems most applicable, and why? Expand the proof accordingly.

There is little to suppose that the elimination rules could be useful, for their premises are different from the propositions on previous lines. This leaves only $\Rightarrow$-I.

```
   From p ⇒ q  infer (p ∧ ¬q) ⇒ ¬p
1 | p ⇒ q                      pr 1
2 | From p ∧ ¬q  infer ¬p
  |   2.1 | p ∧ ¬q             pr 1
  |       |
  |   2.2 | ¬p                 Why?
3 | (p ∧ ¬q) ⇒ ¬p              ⇒-I, 2
```

What can be derived from the propositions appearing on lines previous to 2.2?

Using $\land$-E, we can derive p and $\neg q$ from premise $p \land \neg q$. We then see that q can be derived from $p \Rightarrow q$ and p. (Is it strange that both q and $\neg q$ can be derived?) Keeping these in mind, list the inference rules that could be used to substantiate line 2.2.

Possible inference rules are $\neg$-I, $\land$-E, $\lor$-E, $\neg$-E and $\Rightarrow$-I. Choose the rule that is most applicable and expand the proof accordingly.

The elimination rules don't seem useful here; elimination of **imp** on line 1 results in q, and we already know that $\land$-E can be used to derive p and $\neg q$ from $p \land \neg q$. Only $\neg$-I seems helpful:

```
   From p ⇒ q  infer (p ∧ ¬q) ⇒ ¬p
1 | p ⇒ q                              pr 1
2 | From p ∧ ¬q  infer ¬p
  |  2.1 | p ∧ ¬q                      pr 1
  |  2.2 | From p  infer e ∧ ¬e        (which e?)
  |      |  2.2.1 | p                  pr 1
  |      |        |
  |      |  2.2.2 | e ∧ ¬e             Why?
  |  2.3 | ¬p                          ¬-I, 2.2
3 | (p ∧ ¬q) ⇒ ¬p                      ⇒-I, 2
```

What proposition e should be used on lines 2.2 and 2.2.2? To make the choice, look at the propositions that occur on lines previous to 2.2 and the propositions we know we can derive from them. Expand the proof accordingly.

We reasoned above that we could derive both q and $\neg q$, so the obvious choice is $e = q$. We complete the proof as follows:

From $p \Rightarrow q$ **infer** $(p \wedge \neg q) \Rightarrow \neg p$		
1	$p \Rightarrow q$	pr 1
2	**From** $p \wedge \neg q$ **infer** $\neg p$	
2.1	$p \wedge \neg q$	pr 1
2.2	p	$\wedge$-E, 2.1
2.3	$\neg q$	$\wedge$-E, 2.1
2.4	**From** p **infer** $q \wedge \neg q$	
2.4.1	q	$\Rightarrow$-E, 1, 2.2
2.4.2	$q \wedge \neg q$	$\wedge$-I, 2.4.1, 2.3
2.5	$\neg p$	$\neg$-I, 2.4
3	$(p \wedge \neg q) \Rightarrow \neg p$	$\Rightarrow$-I, 2

The Development of a second proof

Problem. Prove that from $\neg p = q$ follows $\neg(p = q)$. Draw the outline of the proof and fill in the obvious details.

From $\neg p = q$ **infer** $\neg(p = q)$		
1	$\neg p = q$	pr 1
2	$\neg(p = q)$	Why?

What information can be gleaned from the premises?

Rule $=$-E can be used to derive two implications. This seems useful here, since implications will be needed to derive the goal, and we derive both.

From $\neg p = q$ **infer** $\neg(p = q)$		
1	$\neg p \Rightarrow q$	$=$-E, pr 1
2	$q \Rightarrow \neg p$	$=$-E, pr 1
3	$\neg(p = q)$	Why?

The following rules could be used to substantiate line 3: $\neg$-I, $\wedge$-E, $\vee$-E, $\neg$-E and $\Rightarrow$-E. Choose the most likely one and expand the proof accordingly.

The elimination rules don't seem helpful at all, because the premises that would be needed in order to use them are not available and don't seem easy to derive. The *only* rule to try at this point is $\neg$-I —we have little choice!

From $\neg p = q$ **infer** $\neg(p = q)$		
1	$\neg p \Rightarrow q$	$=$-E, pr 1
2	$q \Rightarrow \neg p$	$=$-E, pr 1
3	**From** $p = q$ **infer** $e \wedge \neg e$ (which e?)	
3.1	$p = q$	pr 1
3.2	$e \wedge \neg e$	Why?
4	$\neg(p = q)$	$\neg$-I, 3

What proposition e should be used on lines 3 and 3.2, and how should it be proved? Expand the proof accordingly.

The propositions $\neg p \Rightarrow q$ and $q \Rightarrow \neg p$ are available. In addition, from line 3.1 $p \Rightarrow q$ and $q \Rightarrow p$ can be derived. Let's rearrange these as follows:

$p \Rightarrow q,\ q \Rightarrow \neg p,\ q \Rightarrow p,$ and
$\neg p \Rightarrow q,\ q \Rightarrow \neg p,\ q \Rightarrow p.$

If we assume p we can prove both p and $\neg p$; if we assume $\neg p$ we can also prove p and $\neg p$. Hence we should be able to prove the contradiction $p \wedge \neg p$. So try $e = p$ and write the following proof.

```
From ¬p = q  infer ¬(p = q)
1 | ¬p ⇒ q            =-E, pr 1
2 | q ⇒ ¬p            =-E, pr 1
3 | From p = q  infer p ∧ ¬p
  |  3.1 | p ⇒ q       =-E, pr 1
  |  3.2 | q ⇒ p       =-E, pr 1
  |  3.3 | p           Why?
  |  3.4 | ¬p          Why?
  |  3.5 | p ∧ ¬p      ∧-I, 3.3, 3.4
4 | ¬(p = q)          ¬-I, 3
```

So we are left with concluding the two propositions p and $\neg p$. These are quite simple, using the above reasoning, so let us just show the final proof.

```
From ¬p = q  infer ¬(p = q)
1 | ¬p ⇒ q                          =-E, pr 1
2 | q ⇒ ¬p                          =-E, pr 1
3 | From p = q  infer p ∧ ¬p
  |  3.1 | p ⇒ q                    =-E, pr 1
  |  3.2 | q ⇒ p                    =-E, pr 1
  |  3.3 | From ¬p  infer p ∧ ¬p
  |      |  3.3.1 | q               ⇒-E, 1, pr 1
  |      |  3.3.2 | p               ⇒-I, 3.2, 3.3.1
  |      |  3.3.3 | p ∧ ¬p          ∧-I, 3.3.2, pr 1
  |  3.4 | p                        ¬-E, 3.3
  |  3.5 | From p  infer p ∧ ¬p
  |      |  3.5.1 | q               ⇒-E, 3.1, pr 1
  |      |  3.5.2 | ¬p              ⇒-I, 2, 3.5.1
  |      |  3.5.3 | p ∧ ¬p          ∧-I, pr 1, 3.5.2
  |  3.6 | ¬p                       ¬-I, 3.5
  |  3.7 | p ∧ ¬p                   ∧-I, 3.4, 3.6
5 | ¬(p = q)                        ¬-I, 2
```

At each step of the development of the proof there was little choice. The crucial —and most difficult— point of the development was the choice of inference rule ¬-I to substantiate the last line of the proof, but careful study of the inference rules led to it as the *only* likely candidate. Thus, directed study of the available information can lead quite simply to the proof.

The Tardy Bus Problem

The Tardy Bus Problem is taken from *WFF'N PROOF: The Game of Modern Logic* [1].

THE TARDY BUS PROBLEM. Given are the following premises:

1. If Bill takes the bus, then Bill misses his appointment, if the bus is late.
2. Bill shouldn't go home, if (a) Bill misses his appointment, and (b) Bill feels downcast.
3. If Bill doesn't get the job, then (a) Bill feels downcast, and (b) Bill shouldn't go home.

Which of the following conjectures are true? That is, which can be validly proved from the premises? Give proofs of the true conjectures and counterexamples for the others.

1. If Bill takes the bus, then Bill does get the job, if the bus is late.
2. Bill gets the job, if (a) Bill misses his appointment, and (b) Bill should go home.
3. If the bus is late, then (a) Bill doesn't take the bus, or Bill doesn't miss his appointment, if (b) Bill doesn't get the job.
4. Bill doesn't take the bus if, (a) the bus is late, and (b) Bill doesn't get the job.
5. If Bill doesn't miss his appointment, then (a) Bill shouldn't go home, and (b) Bill doesn't get the job.
6. Bill feels downcast, if (a) the bus is late, or (b) Bill misses his appointment.
7. If Bill does get the job, then (a) Bill doesn't feel downcast, or (b) Bill shouldn't go home.
8. If (a) Bill should go home, and Bill takes the bus, then (b) Bill doesn't feel downcast, if the bus is late.

This problem is typical of the puzzles one comes across from time to time. Most people are confused by them —they just don't know how to deal with them effectively and are amazed at those that do. It turns out, however, that knowledge of propositional calculus makes the problem fairly easy.

The first step in solving the problem is to translate the premises into propositional form. Let the identifiers and their interpretations be:

tb:	Bill *t*akes the *b*us
ma:	Bill *m*isses his *a*ppointment
bl:	The *b*us is *l*ate
gh:	Bill should *g*o *h*ome
fd:	Bill *f*eels *d*owncast
gj:	Bill *g*ets the *j*ob.

The premises are given below. Each has been put in the form of an implication and in the form of a disjunction, knowing that the disjunctive form is often helpful.

Premise 1.	$tb \Rightarrow (bl \Rightarrow ma)$	or	$\neg tb \vee \neg bl \vee ma$
Premise 2.	$(ma \wedge fd) \Rightarrow \neg gh$	or	$\neg ma \vee \neg fd \vee \neg gh$
Premise 3.	$\neg gj \Rightarrow (fd \wedge \neg gh)$	or	$gj \vee (fd \wedge \neg gh)$

Now let's solve the first few problems. In order to save space, Premises 1, 2 and 3 are not written in every proof, but are simply referred to as Premises 1, 2 and 3. Included, however, are propositions derived from them in order to get more true propositions from which to conclude the result.

Conjecture 1: If Bill takes the bus, then Bill does get the job, if the bus is late. Translate the conjecture into propositional form.

In propositional form, the conjecture is $tb \Rightarrow (bl \Rightarrow gj)$. We try to prove "**From** tb **infer** $bl \Rightarrow gj$", which would prove that the conjecture is true. Write the outline for the proof and fill in the obvious details.

```
From tb infer bl ⇒ gj
1 | tb        pr 1
  |
2 | bl ⇒ gj   Why?
```

What propositions can be derived from line 1 and Premises 1, 2 and 3? Expand the proof accordingly.

Proposition $bl \Rightarrow ma$ can be derived from Premise 1 and line 1:

```
From tb infer bl ⇒ gj
1 | tb        pr 1
2 | bl ⇒ ma   ⇒-E, Premise 1, 1
  |
3 | bl ⇒ gj   Why?
```

Which rules could be used to substantiate line 3?

Proposition $bl \Rightarrow gj$ could be an instance of the conclusion of rules $\Rightarrow$-I, $\wedge$-E, $\vee$-E, $\neg$-E, $=$-E and $\Rightarrow$-E. Which seems most useful here? Expand the proof accordingly.

The necessary propositions for the use of the elimination rules are not available, so try $\Rightarrow$-I:

```
From tb infer bl ⇒ gj
1 | tb            pr 1
2 | bl ⇒ ma       ⇒-E, Premise 1, 1
3 | From bl infer gj
  |   3.1 | bl    pr 1
  |       |
  |   3.2 | gj    Why?
4 | bl ⇒ gj       ⇒-I, 3
```

Can any propositions be inferred at line 3.2 from the propositions on previous lines and Premises 1, 2 and 3? Expand the proof accordingly.

Proposition ma can be derived from lines 2 and 3.1:

```
From tb infer bl ⇒ gj
1 | tb            pr 1
2 | bl ⇒ ma       ⇒-E, Premise 1, 1
3 | From bl infer gj
  |   3.1 | bl    pr 1
  |   3.2 | ma    ⇒-E, 2, 3.1
  |       |
  |   3.3 | gj    Why?
4 | bl ⇒ gj       ⇒-I, 3
```

What rules could be used to substantiate line 3.3?

Proposition gj could be an instance of the conclusion of rules $\wedge$-E, $\vee$-E, $\neg$-E and $\Rightarrow$-E. Which ones seem helpful here?

None of the the rules seem helpful. The only proposition available that contains gj is Premise 3, and its disjunctive form indicates that gj must necessarily be true only in states in which $(fd \wedge \neg gh)$ is false (according to theorem (3.4.6)). But there is nothing in Premise 2, the only other place fd and gh appear, to make us believe that $fd \wedge \neg gh$ must be false. Perhaps the conjecture is false. What counterexample —i.e. state in which the conjecture is false— does the structure of the proof and this argument lead to?

Up to line 3.2 of the proof we have assumed or proved $tb = T$, $bl = T$ and $ma = T$. To contradict the conjecture, we need $gj = F$. Finally, the above argument indicates we should try to let $fd \wedge \neg gh$ be true, so we try $fd = T$ and $gh = F$. Indeed, in this state Premises 1, 2 and 3 are true and the conjecture is false.

Conjecture 2: Bill gets the job, if (a) Bill misses his appointment and (b) Bill should go home. Translate the conjecture into propositional form.

This conjecture can be translated as $(ma \wedge gh) \Rightarrow gj$. To prove it we need to prove "**From** $ma \wedge gh$ **infer** gj". Draw the outline of a proof and fill in the obvious details.

From $ma \wedge gh$ **infer** gj

1	$ma \wedge gh$	pr 1
2	gj	Why?

What can we derive from line 1 and Premises 1, 2 and 3? Expand the proof accordingly.

Both line 1 and Premise 2 contain ma and gh. Premise 2 can be put in the form $\neg(ma \wedge gh) \vee \neg fd$. Since $ma \wedge gh$ is on line 1, theorem (3.4.6) together with the law of Negation allows us to conclude that $\neg fd$ is true, or that fd is false. Putting this argument into the proof yields

From $ma \wedge gh$ **infer** gj

1	$ma \wedge gh$	pr 1
2	$\neg(ma \wedge gh) \vee \neg fd$	subs, De Morgan, Premise 2
3	$\neg\neg(ma \wedge gh)$	subs, Negation, 1
4	$\neg fd$	(3.4.6), 2, 1
5	gj	Why?

What inference rule should be used to substantiate line 5? Expand the proof accordingly.

The applicable rules are $\wedge$-E, $\vee$-E, $\neg$-E and $\Rightarrow$-E. This means that an earlier proposition must be broken apart to derive gj. The one that contains gj is Premise 3, and in its disjunctive form it looks promising. To show that gj is true, we need only show that $fd \wedge \neg gh$ is false. But we already know that fd is false, so that we can complete the proof as follows.

From $ma \wedge gh$ **infer** gj

1	$ma \wedge gh$	pr 1
2	$\neg(ma \wedge gh) \vee \neg fd$	subs, De Morgan, Premise 2
3	$\neg\neg(ma \wedge gh)$	subs, Negation, 1
4	$\neg fd$	(3.4.6), 2, 1
5	$\neg fd \vee \neg\neg gh$	$\vee$-I, 4
6	$\neg(fd \wedge \neg gh)$	subs, De Morgan, 5
7	gj	(3.4.6), Premise 3, 6

Conjecture 3: If the bus is late, then (a) Bill doesn't take the bus, or Bill doesn't miss his appointment, if (b) Bill doesn't get the job. Translate the conjecture into propositional form.

Is this conjecture ambiguous? Two possible translations are

$$bl \Rightarrow (\neg gj \Rightarrow (\neg tb \vee \neg ma)), \text{ and}$$
$$bl \Rightarrow (\neg tb \vee (\neg gj \Rightarrow \neg ma))$$

Let us assume the first proposition is intended. It is true if we can prove "**From** bl **infer** $\neg gj \Rightarrow (\neg tb \vee \neg ma)$". Draw the outline of the proof and fill in the obvious details.

	From bl **infer** $\neg gj \Rightarrow (\neg tb \vee \neg ma)$	
1	bl	pr 1
2	$\neg gj \Rightarrow (\neg tb \vee \neg ma)$	Why?

What propositions can be derived from line 1 and the Premises?

No propositions can be derived, at least easily, so let's proceed to the next step. What rule should be used to substantiate line 2? Expand the proof accordingly.

Quite obviously, rule $\Rightarrow$-I should be tried:

	From bl **infer** $\neg gj \Rightarrow (\neg tb \vee \neg ma)$	
1	bl	pr 1
2	**From** $\neg gj$ **infer** $\neg tb \vee \neg ma$	
2.1	$\neg gj$	pr 1
2.2	$\neg tb \vee \neg ma$	Why?
3	$\neg gj \Rightarrow (\neg tb \vee \neg ma)$	

Just before line 2.2, what propositions can be inferred from earlier propositions and Premises 1, 2 and 3? Expand the proof accordingly.

The antecedent of Premise 3 is true, so we can conclude that the consequent is also true:

	From bl **infer** $\neg gj \Rightarrow (\neg tb \vee \neg ma)$	
1	bl	pr 1
2	**From** $\neg gj$ **infer** $\neg tb \vee \neg ma$	
2.1	$\neg gj$	pr 1
2.2	$fd \wedge \neg gh$	$\Rightarrow$-E, Premise 3, 2.1
2.3	fd	$\wedge$-E, 2.2
2.4	$\neg gh$	$\wedge$-E, 2.2
2.5	$\neg tb \vee \neg ma$	Why?
3	$\neg gj \Rightarrow (\neg tb \vee \neg ma)$	

What inference rule should be used to substantiate line 2.5? Expand the proof accordingly.

The proposition on line 2.5 could have the form of the conclusion of rules $\vee$-I, $\wedge$-E, $\vee$-E, $\neg$-E and $\Rightarrow$-E. The first rule to try is $\vee$-I. Its use would require proving that one of $\neg tb$ and $\neg ma$ is true. But, looking at the Premises, this seems difficult. For from Premise 1 we see that both tb and ma could be true, while the other premises are true also because both their conclusions are true. Perhaps there is a contradiction. What is it?

In a state with $tb = T$, $ma = T$, $bl = T$, $gh = F$, $fd = T$ and $gj = F$ Premises 1, 2 and 3 are true, but the conjecture is false.

Exercises for Section 3.5

1. Prove or disprove conjectures 4-8 of the Tardy Bus problem.

2. For comparison, prove the valid conjectures of the Tardy Bus problem using only the equivalence transformation system of chapter 2, and then again in English.

Chapter 4
Predicates

In section 1.3, a state was defined as a function from identifiers to the set of values $\{T, F\}$. The notion of a state is now extended to allow identifiers to be associated with other values, e.g. integers, sequences of characters, and sets. The notion of a proposition will then be generalized in two ways:

1. In a proposition, an identifier may be replaced by any expression (e.g. $x \leqslant y$) that has the value T or F.
2. The quantifiers $\boldsymbol{E}$, meaning "there exists"; $\boldsymbol{A}$, meaning "for all"; and $\boldsymbol{N}$, meaning "number of", are introduced. This requires an explanation of the notions of free identifier and bound identifier and a careful discussion of scope of identifiers in expressions.

Expressions resulting from these generalizations are called *predicates*, and the addition to a formal system (like the system of chapter 2 or 3) of inference rules to deal with them yields a *predicate calculus.*

4.1 Extending the Range of a State

We now consider a state to be a function from identifiers to values, where these values may be other than T and F. In any given context, an identifier has a type, such as *Boolean*, which defines the set of values with which it may be associated. The notations used to indicate the standard types required later are:

Boolean(i): identifier i can be associated (only) with T or F.

natural number(i): i can be associated with a member of $\{0, 1, 2, \cdots\}$.

integer(i): i can be associated with an integer —a member of $\{\cdots, -2, -1, 0, 1, 2, \cdots\}$.

integerset(i): i can be associated with a set of integers.

Other types will be introduced where necessary.

Let P be the expression $x < y$, where x and y have type *integer*. When evaluated, P yields either T or F, so it may replace any identifier in a proposition. For example, replacing b in $(b \wedge c) \vee d$ by P yields

$$((x < y) \wedge c) \vee d.$$

The new assertions like P are called *atomic expressions*, while an expression that results from replacing an identifier by an atomic expression is called a *predicate*. We will not go into detail about the syntax of atomic expressions; instead we will use conventional mathematical notation and rely on the reader's knowledge of mathematics and programming. For example, any expression of a programming language that yields a Boolean result is an acceptable atomic expression. Thus, the following are valid predicates:

$$((x \leqslant y) \wedge (y < z)) \vee (x + y < z)$$
$$(x \leqslant y \wedge y < z) \vee x + y < z$$

The second example illustrates that parentheses are not always needed to isolate the atomic expressions from the rest of a predicate. The precedences of operators in a predicate follow conventional mathematics. For example, the Boolean operators $\wedge$, $\vee$, and $\Rightarrow$ have lower precedence than the arithmetic and relational operators. We will use parentheses to make the precedence of operations explicit where necessary.

Evaluating predicates

Evaluating a predicate in a state is similar to evaluating a proposition. All identifiers are replaced by their values in the state, the atomic expressions are evaluated and replaced by their values (T or F), and the resulting constant proposition is evaluated. For example, the predicate $x < y \vee b$ in the state $\{(x, 2), (y, 3), (b, F)\}$ has the value of $2 < 3 \vee F$, which is equivalent to $T \vee F$, which is T.

Using our earlier notation $s(e)$ to represent the value of expression e in state s, and writing a state as the set of pairs it contains, we show the evaluation of three predicates:

$s((x \leqslant y \wedge y < z) \vee (x+y < z))$ where $s = \{(x,1),(y,3),(z,5)\}$
$= (1 \leqslant 3 \wedge 3 < 5) \vee (1+3 < 5)$
$= (T \wedge T) \vee T$
$= T.$
$s((x \leqslant y \wedge y < z) \vee (x+y < z))$ where $s = \{(x,3),(y,1),(z,5)\}$
$= (3 \leqslant 1 \wedge 1 < 5) \vee (3+1 < 5)$
$= (F \wedge T) \vee T$
$= T.$
$s((x \leqslant y \wedge y < z) \vee (x+y < z))$ where $s = \{(x,5),(y,1),(z,3)\}$
$= (5 \leqslant 1 \wedge 1 < 3) \vee (5+1 < 3)$
$= (F \wedge T) \vee F$
$= F.$

Reasoning about atomic expressions

Just as inference rules were developed for reasoning with propositions, so they should be developed to deal with atomic expressions. For example, we should be able to prove formally that $i < k$ follows from $(i < j \wedge j < k)$. We shall not do this here; as they say, "it is beyond the scope of this book." Instead, we rely on the reader's knowledge of mathematics and programming to reason, as he always has done, about the atomic expressions within predicates.

As mentioned earlier, we will be using expressions dealing with integer arithmetic, real arithmetic (though rarely) and sets. The operators we will be using in these expressions are described in Appendix 2.

The operators **cand** *and* **cor**

Every proposition is well-defined in any state in which all its identifiers have one of the values T and F. When we introduce other types of values and expressions, however, the possibility of undefined expressions (in some states) arises. For example, the expression x/y is undefined if y is 0. We should, of course, be sure that an expression in a program is well-defined in each state in which it will be evaluated, but at times it is useful to allow *part* of an expression to be undefined.

Consider, for example, the expression

$$y = 0 \vee (x/y = 5) .$$

Formally, this expression is undefined if $y = 0$, because x/y is undefined if $y = 0$ and **or** is itself defined only when its operands are T or F. And yet some would argue that the expression should have a meaning in any state where $y = 0$. Since in such states the first operand of **or** is true, and since **or** is defined to be true if either of its operands is true, the

expression should be true. Furthermore, such an interpretation would be quite useful in programming, for it would allow us to say many things more clearly and compactly. For example, consider being able to write

if $y = 0 \vee (x/y = 5)$ **then** *s1* **else** *s2*

as opposed to

```
if y = 0 then s1
    else if x/y = 5 then s1
                    else s2
```

Rather than change the definition of **and** and **or**, which would require us to change our formal logic completely, we introduce two new operators: **cand** (for conditional **and**) and **cor** (for conditional **or**). The operands of these new operators can be any of *three* values: F, T and U (for Undefined). The new operators are defined by the following truth table.

b	c	b **cand** c	b **cor** c
T	T	T	T
T	F	F	T
T	U	U	T
F	T	F	T
F	F	F	F

b	c	b **cand** c	b **cor** c
F	U	F	U
U	T	U	U
U	F	U	U
U	U	U	U

This definition says nothing about the *order* in which the operands should be evaluated. But the intelligent way to evaluate these operations, at least on current computers, is in terms of the following equivalent conditional expressions:

b **cand** c: **if** b **then** c **else** F
b **cor** c: **if** b **then** T **else** c

Operators **cand** and **cor** are not commutative. For example, b **cand** c is not equivalent to c **cand** b. Hence, care must be exercised in manipulating expressions containing them. The following laws of equivalence *do* hold for **cand** and **cor** (see exercise 5). These laws are numbered to correspond to the numbering of the laws in chapter 2.

2. **Associativity**: *E1* **cand** (*E2* **cand** *E3*) = (*E1* **cand** *E2*) **cand** *E3*
   E1 **cor** (*E2* **cor** *E3*) = (*E1* **cor** *E2*) **cor** *E3*

3. **Distributivity**:
   E1 **cand** (*E2* **cor** *E3*) = (*E1* **cand** *E2*) **cor** (*E1* **cand** *E3*)
   E1 **cor** (*E2* **cand** *E3*) = (*E1* **cor** *E2*) **cand** (*E1* **cor** *E3*)

4. **De Morgan**: $\neg(E1$ **cand** $E2) = \neg E1$ **cor** $\neg E2)$
$\neg(E1$ **cor** $E2) = \neg E1$ **cand** $\neg E2)$

6. **Excluded Middle**: $E1$ **cor** $\neg E1 = T$ (provided $E1$ is well-defined)

7. **Contradiction**: $E1$ **cand** $\neg E1 = F$ (provided $E1$ is well-defined)

10. **cor-simplification**
$E1$ **cor** $E1 = E1$
$E1$ **cor** $T = T$ (provided $E1$ is well-defined)
$E1$ **cor** $F = E1$
$E1$ **cor** $(E1$ **cand** $E2) = E1$

11. **cand-simplification**
$E1$ **cand** $E1 = E1$
$E1$ **cand** $T = E1$
$E1$ **cand** $F = F$ (provided $E1$ is well-defined)
$E1$ **cand** $(E1$ **cor** $E2) = E1$

In addition, one can derive various laws that combine **cand** and **cor** with the other operations, for example,

$E1$ **cand** $(E2 \vee E3) = (E1$ **cand** $E2) \vee (E1$ **cand** $E3)$

Further development of such laws are left to the reader.

Exercises for Section 4.1

1. The first two exercises consist of evaluating predicates and other expressions involving integers and sets. Appendix 2 gives more information on the operations used. The state s in which the expressions should be evaluated consists of two integer identifiers x, y, a Boolean identifier b, two set identifiers m, n and an integer array $c[1:3]$. Their values are:

$$x=7,\ y=2,\ b=T,\ m=\{1,2,3,4\},\ n=\{2,4,6\},\ c=(2,4,6)$$

(a) $x \div y = 3$
(b) $(x-1)\div y = 3$
(c) $(x+1)\div y = 3$
(d) $ceil(x/y) = x \div y + 1$
(e) $floor((x+1)/y) = (x+1)\div y$
(f) $floor(-x/y) = -3$
(g) $ceil(x/y) = x \div y$
(h) $-ceil(-x/y) = x \div y$
(i) 7 **mod** 2
(j) $floor(x/y) = x \div y$
(k) $min(floor(x/2), ceil(x/2)) < ceil(x/2)$
(l) $(abs(-x) = -abs(x)) = b$
(m) $b \vee x < y$
(n) 19 **mod** 3

2. Evaluate the following expressions in the state given in exercise 1.

(a) $m \cup n$
(b) $m \cap n$
(c) $x \in m \wedge b$
(d) $m \subset n \wedge b$
(e) $\varnothing \subset m$
(f) $\{i \mid i \in m \wedge even(i)\} \subset n$
(g) $|m| \in m$
(h) $|n| \in n$
(i) $(\{|m|\} \cup \{6,7\}) \subset n$
(j) $|m| + |n| = |m \cup n|$
(k) $min(m)$
(l) $\{i \mid i \in m \wedge i \in n\}$

3. Evaluate the following predicates in the state given in exercise 1. Use U for the value of an undefined expression.

(a) $b \vee x/(y-2) = 0$
(b) b **cor** $x/(y-2) = 0$
(c) $b \wedge x/(y-2) = 0$
(d) b **cand** $x/(y-2) = 0$
(e) $x = 0 \wedge x/(y-2) = 0$
(f) $x = 0$ **cand** $x/(y-2) = 0$
(g) $1 \leqslant y \leqslant 3$ **cand** $c[y] \in m$
(h) $1 \leqslant y \leqslant 3$ **cor** $c[x] \in m$
(i) $1 \leqslant y \leqslant 3$ **cand** $c[y+1] \in m$
(j) $1 \leqslant x \leqslant 3$ **cor** $c[y] \in m$

4. Consider propositions a, b and c as having the values F, T or U (for undefined). Describe all states where the commutative laws a **cor** $b = b$ **cor** a and a **cand** $b = b$ **cand** a do not hold.

5. Prove that the laws of Associativity, Distributivity, De Morgan, Excluded Middle, Contradiction, **cor**-simplification and **cand**-simplification, given just before these exercises, hold. Do this by building a truth table for each one.

4.2 Quantification

Existential quantification

Let m and n be two integer expressions satisfying $m \leqslant n$. Consider the predicate

(4.2.1) $E_m \vee E_{m+1} \vee \cdots \vee E_{n-1}$,

where each E_i is a predicate. (4.2.1) is true in any state in which at least one of the E_i is true. It can be expressed using the *existential quantifier* $\boldsymbol{E}$ (read "there exists") as

(4.2.2) $(\boldsymbol{E}\, i: m \leqslant i < n: E_i)$.

The set of values that satisfy $m \leqslant i < n$ is called the *range* of the *quantified identifier* i. Predicate (4.2.2) is read in English as follows.

$(\boldsymbol{E}\, i$	there exists at least one (integer) i
:	such that
$m \leqslant i < n$	i is between m and $n-1$ (inclusive)
:	for which the following holds:
$E_i)$	E_i.

The reader is no doubt already familiar with some forms of quantification in mathematics. For example,

$$\sum_{i=m}^{n-1} s_i = s_m + s_{m+1} + \cdots + s_{n-1}$$

$$\prod_{i=m}^{n-1} s_i = s_m * s_{m+1} * \cdots * s_{n-1}.$$

stand for the sum and product of the values s_m, s_{m+1}, ..., s_{n-1}, respectively. These can be written in a more linear fashion, similar to (4.2.1), as follows, and we shall continue to use this new form:

$$(\Sigma\, i: m \leqslant i < n: s_i)$$

$$(\Pi\, i: m \leqslant i < n: s_i)$$

At this point, (4.2.2) is simply an abbreviation for (4.2.1). It can be recursively defined as follows:

(4.2.3) **Definition of $\boldsymbol{E}$**:
$(\boldsymbol{E}\, i: m \leqslant i < m: E_i) = F$, and, for $k \geqslant m$,
$(\boldsymbol{E}\, i: m \leqslant i < k+1: E_i) = (\boldsymbol{E}\, i: m \leqslant i < k: E_i) \vee E_k$ □

Remark: The base case of this recursive definition, which concerns an empty range $m \leqslant i < m$ for i, brings out an interesting point. The disjunction of zero predicates, $(\boldsymbol{E}\, i: m \leqslant i < m: E_i)$, has the value F: "oring" 0 predicates together yields a predicate that is always false. For example, the following predicates are equivalent to F:

$(\boldsymbol{E}\, i: 0 \leqslant i < 0: i = i)$
$(\boldsymbol{E}\, i: -3 \leqslant i < -3: T)$

The disjunction of zero disjuncts is F. The conjunction of zero conjuncts turns out to be T. Similarly, the sum of zero values is 0 and the product of zero values is 1. These four facts are expressed as

$(\Sigma\, i: 0 \leqslant i < 0: x_i) = 0$,
$(\Pi\, i: 0 \leqslant i < 0: x_i) = 1$,
$(\boldsymbol{E}\, i: 0 \leqslant i < 0: E_i) = F$,
$(\boldsymbol{A}\, i: 0 \leqslant i < 0: E_i) = T$. (Notation explained subsequently)

The value 0 is called the *identity element* of addition, because any number added to 0 yields that number. Similarly, 1, F and T are the identity elements of the operators $*$, **or** and **and**, respectively. **End of remark**

The following examples use quantification over two identifiers. They are equivalent; they assert the existence of i and j between 1 and 99 such that i is prime and their product is 1079 (is this true?). The third one uses the convention that successive quantifications with the same range, $(\boldsymbol{E}\, i: m \leqslant i < n: (\boldsymbol{E}\, j: m \leqslant j < n: (\boldsymbol{E}\, k: m \leqslant k < n: \cdots)))$ can be written as $(\boldsymbol{E}\, i, j, k: m \leqslant i, j, k < n: \cdots)$.

(1) $(\boldsymbol{E}\, i: 0 \leqslant i < 100: (\boldsymbol{E}\, j: 0 \leqslant j < 100: prime(i) \wedge i*j = 1079))$
(2) $(\boldsymbol{E}\, i: 0 \leqslant i < 100: prime(i) \wedge (\boldsymbol{E}\, j: 0 \leqslant j < 100: i*j = 1079))$
(3) $(\boldsymbol{E}\, i, j: 0 \leqslant i, j < 100: prime(i) \wedge i*j = 1079))$

Universal quantification

The *universal quantifier*, $\boldsymbol{A}$, is read as "for all". The predicate

(4.2.4) $(\boldsymbol{A}\, i: m \leqslant i < n: E_i)$

is true in a state *iff*, for all values i in the range $m \leqslant i < n$, E_i is true in that state.

We now define $\boldsymbol{A}$ in terms of $\boldsymbol{E}$, so that, formally, we need deal only with one of them as a new concept. Predicate (4.2.4) is true *iff* all the E_i are true, so we see that it is equivalent to

$$\begin{array}{lll} & E_m \wedge E_{m+1} \wedge \cdots \wedge E_{n-1} & \\ = & \neg\neg(E_m \wedge E_{m+1} \wedge \cdots \wedge E_{n-1}) & \text{(Negation)} \\ = & \neg(\neg E_m \vee \neg E_{m+1} \vee \cdots \vee \neg E_{n-1}) & \text{(De Morgan)} \\ = & \neg(\boldsymbol{E}\, i: m \leqslant i < n: \neg E_i) & \end{array}$$

This leads us to define (4.2.4) as

(4.2.5) **Definition.** $(\boldsymbol{A}\, i: m \leqslant i < n: E_i) = \neg(\boldsymbol{E}\, i: m \leqslant i < n: \neg E_i)$. □

Now we can prove that (4.2.4) is true if its range is empty:

$$\begin{array}{lll} & (\boldsymbol{A}\, i: m \leqslant i < m: E_i) & \\ = & \neg(\boldsymbol{E}\, v: m \leqslant i < m: \neg E_i) & \\ = & \neg F & \text{(because the range of } \boldsymbol{E} \text{ is empty)} \\ = & T & \end{array}$$

Numerical quantification

Consider predicates E_0, E_1, ... It is quite easy to assert formally that k is the smallest integer such that E_k holds. We need only indicate that E_0 through E_{k-1} are false and that E_k is true: