Chapter 1

Propositional Logic

1.1 Introduction

A *proposition* is a statement which is either true or false.

Examples 1.1.1
"There are infinitely many primes"; "$5 > 3$"; and "14 is a square number" are propositions. A statement like "x is odd," however, is *not* a proposition (but it becomes one when x is substituted by a particular number).

A *propositional connective* is a way of combining propositions so that the truth or falsity of the compound proposition depends only on the truth or falsity of the components. The most common connectives are:

connective	*symbol*
not (negation)	$\neg$
and (conjunction)	$\wedge$
or (disjunction)	$\vee$
implies (implication)	$\Rightarrow$
iff (equivalence)	$\Leftrightarrow$

Notice that each connective has a symbol which is traditionally used to represent it. This way we can distinguish between the desired precise mathematical meaning of the connective (e.g. $\neg$) and the possibly vague or ambiguous meaning of an english word (e.g. "implies"). The intended mathematical meanings of the connectives are as follows:

- “Not” ($\neg$) is the only common *unary connective*: it applies to only one proposition, and negates its truth value. The others are *binary connectives* that apply to two propositions:
- A proposition combined with “and” ($\wedge$) is true only if *both* of its components are true, while…
- A proposition combined with “or” ($\vee$) is true whenever *either or both* of its components are true. That is, $\vee$ is the so-called “inclusive or”.
- The “iff” (short for for “if and only if”) or equivalence connective ($\Leftrightarrow$) is true whenever both of its components have the same truth value: either both true or both false.
- The “implication” connective is the only common binary connective which is not symmetric: $p \Rightarrow q$ is not the same as $q \Rightarrow p$. $p \Rightarrow q$ is intended to convey the meaning that whenever p is true, q must also be. So if p is true, $p \Rightarrow q$ is true if and only if q is also true. If, on the other hand, p is false, we say that $p \Rightarrow q$ is “vacuously” true. The statement that $p \Rightarrow q$ does not carry any “causal” connotation, unlike the English phrases we use (for lack of anything better) to pronounce it (such as “p implies q” or “if p, then q”).

We can summarize the meanings of these connectives using *truth tables*:

p	$\neg p$
T	F
F	T

p	q	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \Leftrightarrow q$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

Examples 1.1.2
“$\neg 2 = 2$” is false;
“$25 = 5^2 \wedge 6 > 5$” is true;
“5 is even $\vee$ 16 is a square” is true;
“5 is odd $\vee$ 16 is a square” is true;

“(the square of an odd number is odd)⇒(25 is odd)” is true;
“(the square of an odd number is even)⇒(25 is odd)” is true;
“(the square of an odd number is odd)⇒(10 is odd)”is false.

There are many different ways to express a compound statement, and corresponding to these there are different ways to combine propositions using these connectives which are *equivalent*: their truth values are the same for any truth values of the original propositions. Here are some examples:

(i) The statements “The cafeteria either has no pasta, or it has no sauce” and “The cafeteria doesn't have both pasta and sauce” are two ways to say the same thing. If $p =$ “The cafeteria has pasta” and $q =$ “The cafeteria has sauce”, then this means that

$$(\neg p) \vee (\neg q) \quad \text{and} \quad \neg(p \wedge q)$$

are equivalent. This is one of what are known as *De Morgan's Laws*.

(ii) The *law of the double negative* says that $\neg\neg p$ and p are equivalent. That is, if something is not false, it is true, and vice versa.

(iii) The statements “If it rained this morning, the grass is wet” and “If the grass is dry, it didn't rain this morning” are two more ways to say the same thing. Now if $p =$ “It rained this morning” and $q =$ “The grass is wet”, then this means that

$$(p \Rightarrow q) \quad \text{and} \quad (\neg q) \Rightarrow (\neg p)$$

are equivalent. The latter statement is called the *contrapositive* of the first. Note that because of the law of the double negative, the first statement is also equivalent to the contrapositive of the second.

(iv) Yet another way to say the same thing is “Either it did not rain this morning, or the grass is wet.” That is, another proposition equivalent to those above is

$$(\neg p) \vee q.$$

One consequence of this is that we could, if we wanted to, do without the $\Rightarrow$ connective altogether. We will see more about this in section 1.6.B.

(v) Two more related statements which are *not* equivalent to $p \Rightarrow q$ are the *converse*, $q \Rightarrow p$ ("If the grass is wet, it rained this morning"), and the *inverse*, $(\neg p) \Rightarrow (\neg q)$ ("If it didn't rain this morning, the grass isn't wet"). There might, for example, be sprinklers that keep the grass wet even when it doesn't rain. The converse and inverse are contrapositives of each other, however, so they have the same truth value.

In section 1.5 we will revisit this notion of "equivalence" in a formal context.

1.2 Syntax of propositional logic

In order to rigorously prove results about the structure and truth of propositions and their connectives, we must set up a mathematical structure for them. We do this with a *formal language*, the language of *propositional logic*, which we will work with for the rest of the chapter.

1.2.A The Language of Propositional Logic

Definition 1.2.1 A *formal language* consists of a set of symbols together with a set of rules for forming "grammatically correct" strings of symbols in this language.

Definition 1.2.2 In the *language of propositional logic* we have the following list of symbols:

(i) *propositional variables*: this is an infinite list $p_1, p_2, p_2, \ldots$ of symbols. We often use $p, q, r, \ldots$ to denote propositional variables.

(ii) *symbols for the (common) propositional connectives*: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$.

(iii) *parentheses*: (,).

Definition 1.2.3 A *string* or *word* in a formal language is any finite sequence of the symbols in the language. We include in this the *empty string* containing no symbols at all.

Example 1.2.4 The following are examples of strings in the language of propositional logic:

$$p\vee,\ pq\Rightarrow,\ pqr,\ (p\wedge q),\ p)$$

Definition 1.2.5 If $S = s_1 \dots s_n$ is a string with n symbols, we call n the *length* of S and denote it by $|S|$. The length of the empty string is 0.

We will next specify the rules for forming "grammatically correct" strings in propositional logic, which we call *well-formed formulas* (*wff*s) or just *formulas.*

Definition 1.2.6 (Well-Formed Formulas)

(i) Every propositional variable p is a wff.

(ii) If A is a wff, so is $\neg A$.

(iii) If A, B are formulas, so are $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$, and $(A \Leftrightarrow B)$.

Thus a string A is a wff exactly when there is a finite sequence

$$A_1, \dots, A_n$$

(called a *parsing sequence*) such that $A_n = A$ and for each $1 \leq i \leq n$, A_i is either (1) a propositional variable, (2) for some $j < i$, $A_i = \neg A_j$, or (3) for some $j, k < i$, $A_i = (A_j * A_k)$, where $*$ is one of $\wedge, \vee, \Rightarrow, \Leftrightarrow$.

Examples 1.2.7

(i) $(p \Rightarrow (q \Rightarrow r))$ is a wff with parsing sequence

$$p,\ q,\ r,\ (q \Rightarrow r),\ (p \Rightarrow (q \Rightarrow r)).$$

(Also note that

$$q,\ r,\ (q \Rightarrow r),\ p,\ (p \Rightarrow (q \Rightarrow r))$$

is another parsing sequence, so parsing sequences are not unique.)

(ii) $(\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q))$ is a wff with parsing sequence

$$p,\ q,\ (p \wedge q),\ \neg(p \wedge q),\ \neg p,\ \neg q,\ (\neg p \vee \neg q),\ (\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)).$$

(iii) $p \Rightarrow qr$, $)p \Leftrightarrow$, $\vee(q\neg)$, and $p \wedge q \vee r$ are not wff, but to prove this is rather more tricky. We will see some ways to do this in section 1.2.B.

Remark. We have not yet assigned any "meaning" to any of the symbols in our formal language. While we intend to interpret symbols such as $\vee$ and $\Rightarrow$ eventually in a way analogous to the propositional connectives "or" and "implies" in section 1.1, at present they are simply symbols that we are manipulating formally.

1.2.B Induction on Formulas

We can prove properties of formulas by induction on the length of a formula. Suppose $\Phi(A)$ is a property of a formula A. For example, $\Phi(A)$ could mean "A has the same number of left and right parentheses." If $\Phi(A)$ holds for all formulas of length 1 (i.e. the propositional variables) and whenever $\Phi(A)$ holds for all formulas A of length $\leq n$, then it holds for all formulas of length $n+1$, we may conclude, by the principle of mathematical induction, that $\Phi(A)$ holds for *all* formulas.

However, because of the way formulas are defined, it is most useful to use another form of induction, sometimes called *induction on the construction of wff* (or just *induction on formulas*):

Let $\Phi(A)$ be a property of formulas A. Suppose:

(i) *Basis of the induction*: $\Phi(A)$ holds, when A is a propositional variable.

(ii) *Induction step*: If $\Phi(A), \Phi(B)$ hold for two formulas A, B, then

$$\Phi(\neg A), \Phi((A * B))$$

hold as well, where $*$ is one of the binary connectives $\wedge, \vee, \Rightarrow, \Leftrightarrow$.

Then $\Phi(A)$ holds for *all* formulas A. We can prove the validity of this procedure using standard mathematical induction, and the existence of a parsing sequence for any wff.

Example 1.2.8 One can easily prove, using this form of induction, that every formula A has the same number of left and right parentheses. This result can be used to show that certain strings, for example $)p \Leftrightarrow$, are not wffs, because they have unequal numbers of left and right parentheses.

Example 1.2.9 If $s_1 s_2 \ldots s_n$ is any string, then an *initial segment* of it is a string $s_1 \ldots s_m$, where $0 \leq m \leq n$ (when $m = 0$, this means the empty string). If $m < n$ we call this a *proper* initial segment.

Again we can easily prove by induction on the construction of wff that a nonempty proper initial segment of a formula either consists of a string of $\neg$'s or else has more left than right parentheses. (To see some examples, consider $\neg\neg(p \Rightarrow q), ((p \Rightarrow (q \wedge p)) \Rightarrow \neg p)$.) Another easy induction shows that no formula can be empty or just a string of $\neg$'s, so using also the previous example, *no proper initial segment of a wff is a wff.*

This can also be used to show certain strings (for example, $p \Rightarrow qr$) are not wffs, as they contain proper initial segments that are wffs (in the example, p). Similar methods can be used for other non-wffs.

1.2.C Unique Readability

While parsing sequences are not unique, as mentioned in examples 1.2.7, there is a sense in which both alternative parsing sequences offered are "the same": they break the wff down in the same way, to the same components, albeit in a different order. Using induction on formulas, we can prove that this will always be the case. This is called *unique readability.*

Theorem 1.2.10 (Unique Readability Theorem) *Every wff is in exactly one of the forms:*

(i) p *(i.e., a propositional variable);*

(ii) $\neg A$ *for* A *a uniquely determined wff;*

(iii) $(A * B)$, *for uniquely determined wff's* A, B *and uniquely determined* $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$.

Proof. By induction on the construction of formulas, it is straightforward that any formula must be of one of these forms. To prove uniqueness first notice that no wff can be in more than one of the forms (i), (ii), (iii). And obviously no two propositional variables are the same, and if $\neg A = \neg B$, then $A = B$.

So it is enough to show that in case (iii), if $(A * B) = (C \circ D)$, for $*, \circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$, then $A = C, B = D$ and $* = \circ$. First notice that, by eliminating the left parenthesis, we have that $A * B) = C \circ D)$. From this it follows that either A is an initial segment of C or vice versa. If $A \neq C$, then one of A, C is a proper initial segment of the other, which is impossible, as they are both formulas. So we must have $A = C$. Thus $*B) = \circ D)$ and so $* = \circ$ and $B = D$. $\dashv$

Definition 1.2.11 In form (ii) we call $\neg$ the *main connective* and in (iii) we call $*$ the *main connective*. (Thus the main connective is the one applied last.)

Example 1.2.12 In $((p \Rightarrow (q \Rightarrow p)) \Rightarrow q)$ the main connective is the third "$\Rightarrow$" (from left to right).

Given a formula A, it might be hard to recognize immediately which is the main connective. Consider for example the following formula:

$$\begin{aligned}&(((((p \vee \neg q) \vee \neg r) \wedge ((\neg p \vee (s \vee \neg r)) \wedge ((q \vee s) \vee p)))\wedge \\ &\qquad (((\neg q \vee \neg r) \vee s) \wedge (s \vee (\neg r \vee q)))) \wedge ((p \vee (\neg r \vee s)) \wedge (s \vee \neg r)))\end{aligned}$$

(it turns out to be the 5th "$\wedge$"). There is however an easy algorithm for determining the main connective: In the formula $(A * B)$ the main connective $*$ is the first binary connective (from left-to-right) such that in the string preceding it the number of left parentheses is exactly one more than the number of right parentheses.

1.2.D Recursive Definitions

Often we will want to define a function or characteristic of wffs, and the most natural way to do it is to break down the formula into simpler formulas one step at a time. To make this rigorous, we can use the unique readability theorem to justify the following principle of *definition by recursion on the construction of formulas*:

Let X be a set and

$$\begin{aligned} f &: \{p_1, p_2, \dots\} \to X \\ f_\neg &: X \to X \\ f_\wedge, f_\vee, f_\Rightarrow, f_\Leftrightarrow &: X^2 \to X \end{aligned}$$

be given functions. Then there is a unique function g from the set of all formulas into X such that

$$\begin{aligned} g(p) &= f(p) \\ g(\neg A) &= f_\neg(g(A)) \\ g((A * B)) &= f_*(g(A), g(B)), \quad * \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}. \end{aligned}$$

Without unique readability, the existence and uniqueness of g would be called into question: if there were two different ways of breaking down a formula, then the corresponding compositions of the fs might yield different values of g for the same formula. But now that we know unique readibility, g is well-defined and unique.

Examples 1.2.13

(i) Let

$$\begin{aligned} L(p) &= 1 \\ L(\neg A) &= L(A) + 1 \\ L((A * B)) &= L(A) + L(B) + 3. \end{aligned}$$

Then it is easy to see that $L(A) = |A| =$ the length of A.

(ii) Let

$$\begin{aligned} C(P) &= 0 \\ C(\neg A) &= C(A) + 1 \\ C((A * B)) &= C(A) + C(B) + 1. \end{aligned}$$

Then $C(A) =$ the number of connectives in A.

(iii) Let p be a propositional variable and P a formula. We define for each formula A the formula $g(A) = A[p/P]$ as follows:

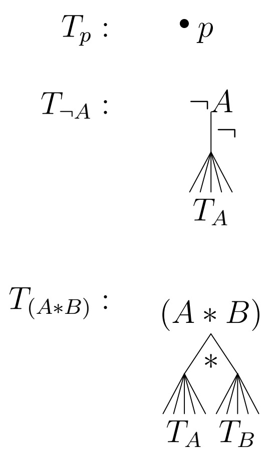
$$\begin{aligned} g(p) &= P, \\ g(q) &= q \quad \text{ if } q \neq p, \\ g(\neg A) &= \neg g(A) \\ g((A * B)) &= (g(A) * g(B)). \end{aligned}$$

Then it is easy to see that $A[p/P]$ is the formula obtained by substituting every occurence of p in A by P.

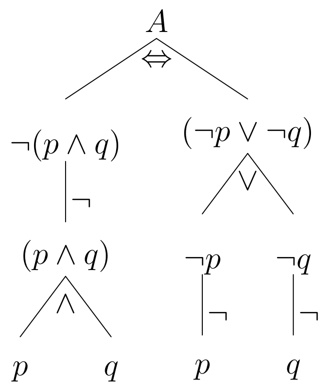
For example, if $A = (\neg p \Rightarrow (q \Rightarrow p))$ and $P = (r \vee q)$, then

$$A[p/P] = (\neg(r \vee q) \Rightarrow (q \Rightarrow (r \vee q))).$$

(iv) The *parse tree* T_A of a formula A is defined recursively as follows:



For example, if $A = (\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q))$, then its parse tree is



The parse tree, starting now from the bottom up, gives us various ways of constructing a parsing sequence for the formula; for example:

$$p,\ q,\ (p \wedge q),\ \neg(p \wedge q),\ \neg p,\ \neg q,\ (\neg p \vee \neg q),\ (\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)).$$

By unique readability, all parsing sequences can be obtained in this manner. Thus the parse tree extracts the "common part" out of each parsing sequence.

(v) The formulas occurring in the parse tree of a formula A are called the *subformulas* of A. These are the formulas that arise in the construction of A. If we let $\mathrm{subf}(A)$ denote the set of subformulas of A, then subf can be defined by the following recursion:

$$\begin{aligned}\mathrm{subf}(p) &= \{p\}\\ \mathrm{subf}(\neg A) &= \mathrm{subf}(A) \cup \{\neg A\}\\ \mathrm{subf}\big((A * B)\big) &= \mathrm{subf}(A) \cup \mathrm{subf}(B) \cup \{(A * B)\}.\end{aligned}$$

1.2.E Recognizing Formulas

We will next discuss an algorithm for verifying that a given string is a formula.

Algorithm 1.2.14 Given a string S, let x be a propositional variable not occurring in S and let $S(x)$ the string we obtain from S by substituting every propositional variable by x.

For any string T containing only the variable x, let T' be the string obtained from T as follows: If T contains no *substring* (i.e. a string contained in T as a consecutive sequence of symbols) of the form $\neg x, (x \wedge x), (x \vee x), (x \Rightarrow x), (x \Leftrightarrow x)$, then $T' = T$. Otherwise substitute the leftmost occurence of any one of these forms by x, to obtain T'. Notice that if $T \neq T'$, then $|T'| < |T|$.

Now starting with an arbitrary string S, form first $S(x)$, and then define recursively $S_0 = S(x)$, $S_{n+1} = S'_n$. This process stops when we hit n_0 for which $S_{n_0+1} = S'_{n_0} = S_{n_0}$, i.e., S_n has no substring of the above form. This must necessarily happen since otherwise we would have $|S_0| > |S_1| > \dots$ (ad infinitum), which is impossible.

We now claim that if $S_{n_0} = x$, then S is a formula, otherwise it is not.

Examples 1.2.15

(i) $((p \Rightarrow (q \vee \neg p) \Rightarrow (r \vee (\neg s \Rightarrow t)))) = S$

$$\begin{array}{ccl}((x \Rightarrow (x \vee \neg x) \Rightarrow (x \vee (\neg x \Rightarrow x)))) &=& S(x) = S_0\\ ((x \Rightarrow (x \vee x) \Rightarrow (x \vee (\neg x \Rightarrow x)))) &=& S_1\\ ((x \Rightarrow x \Rightarrow (x \vee (\neg x \Rightarrow x)))) &=& S_2\\ ((x \Rightarrow x \Rightarrow (x \vee (x \Rightarrow x)))) &=& S_3\\ ((x \Rightarrow x \Rightarrow (x \vee x))) &=& S_4\\ ((x \Rightarrow x \Rightarrow x)) &=& S_5 = S_6\end{array}$$

So S is not a wff.

(ii) $((p \Rightarrow \neg q) \Rightarrow (p \vee (\neg r \wedge s))) = S$

$$\begin{array}{ccl}
((x \Rightarrow \neg x) \Rightarrow (x \vee (\neg x \wedge x))) & = & S(x) = S_0 \\
((x \Rightarrow x) \Rightarrow (x \vee (\neg x \wedge x))) & = & S_1 \\
(x \Rightarrow (x \vee (\neg x \wedge x))) & = & S_2 \\
((x \Rightarrow (x \vee (x \wedge x))) & = & S_3 \\
((x \Rightarrow (x \vee x)) & = & S_4 \\
(x \Rightarrow x) & = & S_5 \\
x & = & S_6 = S_7
\end{array}$$

So S is a wff.

We will now prove the correctness of this algorithm in two parts: first, that it recognizes all wffs, and second, that it recognizes only wffs.

Part I. If $S = A$ is a wff, then this process stops at x.

Proof of part I. First notice that S is a wff iff $S(x)$ is a wff. (This can be easily proved by induction on the construction of formulas.)

So if A is a wff, so is $B = A(x)$. Thus it is enough to show that if we start with any wff B containing only the variable x and we perform this process, we will end up with x. For this it is enough to show that for any such formula B, B' is also a formula. Because then when the process $B_0 = B, B_1, \ldots, B_{n_0}$ stops, we have that B_{n_0} is a formula containing only the variable x and $B'_{n_0} = B_{n_0}$, i.e., B_{n_0} contains no substrings of the form $\neg x, (x \wedge x), (x \vee x), (x \Rightarrow x), (x \Leftrightarrow x)$, therefore it must be equal to x.

We can prove that for any formula B, B' is a formula, by induction on the construction of B. If $B = x$, then $B' = B = x$. If $B = \neg C$, then either $C = x$ and $B' = x$, or else $B' = \neg C'$, since $C \neq x$ implies that C cannot start with x. If $B = (C * D)$, then, unless $C = D = x$, in which case $B' = x$, either $B' = (C' * D)$ or $B' = (C * D')$. $\dashv$

Part II. If, starting from S, this process terminates at x, then S is a formula.

Proof of part II. Again we can assume that we start with a string S_0 containing only the variable x and show that if $S_0, S_1, \ldots, S_{n_0} = S'_{n_0} = x$, for some n_0, where $S_{n+1} = S'_n$, then S_0 is a formula. To see this notice that S_{n_0} is a formula and S_{n-1} is obtained from S_n by substituting some occurrence x by a formula. So working backwards, and using the fact that if in a wff we substitute an occurrence of a variable by a formula we still get a formula (a fact which is easy to prove by induction), we see that $S_{n_0}, S_{n_0-1}, S_{n_0-2}, \ldots, S_1, S_0$ are all formulas. $\dashv$

1.3 Polish and Reverse Polish notation

We will now describe two different ways of writing formulas in which parentheses can be avoided. These methods are often better suited to computer processing, and many computer programs store formulas internally in one of these notations, even if they use standard notation for input and output.

1.3.A Polish Notation

We define a new formal language as follows:

(i) The symbols are the same as before, *except* that there are no parentheses, i.e.:

$$p_1, p_2, \ldots, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$$

(ii) We have the following rules for forming grammatically correct formulas, which we now call *Polish wff* or just *P-wff* or *P-formulas*:

(a) Every propositional variable p is a P-wff;

(b) If A is a P-wff, so is $\neg A$.

(c) If A, B are P-wff, so is $*AB$, where $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$

Remark. Polish notation is named after the Polish logician Łukasiewicz who first introduced it, although he used N, K, A, C, and E in place of $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$ respectively.

Example 1.3.1

$$\Rightarrow\Rightarrow s\neg t\neg \vee p \wedge \neg qs$$

is a P-wff as the following parsing sequence demonstrates

q, $\neg q$, s, $\wedge\neg qs$, p, $\vee p \wedge \neg qs$, $\neg \vee p \wedge \neg qs$, t, $\neg t$, $\Rightarrow s\neg t$, $\Rightarrow\Rightarrow s\neg t\neg \vee p \wedge \neg qs$.

In ordinary notation this P-wff corresponds to the wff:

$$((s \Rightarrow \neg t) \Rightarrow \neg(p \vee (\neg q \wedge s))).$$

As with regular formulas, we have unique readability.

Theorem 1.3.2 (Unique Readability of P-wff) *Every P-wff A is in exactly one of the forms,*

(i) p

(ii) $\neg B$

(iii) $*BC$*, where* $* \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$*, for uniquely determined P-wff* B, C*.*

Proof. Clearly the only thing we have to prove is that if B, C, B', C' are P-wff and $BC = B'C'$, then $B = B', C = C'$. If $BC = B'C'$ then one of B, B' is an initial segment of the other, so it is enough to prove the following lemma:

Lemma 1.3.3 *No nonempty proper initial segment of a P-wff is a P-wff.*

Proof of Lemma 1.3.3. We define a *weight* for each symbol as follows:

$$\begin{aligned} w(p) &= 1 \\ w(\neg) &= 0 \\ w(*) &= -1 \quad \text{if } * \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}. \end{aligned}$$

Then we define the weight of any string $S = s_1 s_2 \ldots s_n$, by

$$w(S) = \sum_{i=1}^{n} w(s_i).$$

For example, $w(\wedge \Rightarrow s \neg pq) = 1$. It is easy to show by induction on the construction of a P-wff that if A is a P-wff, then $w(A) = 1$.

We now define a *terminal segment* of a string $S = s_1 \ldots s_n$ to be any string of the form

$$s_m s_{m+1} \ldots s_n \quad (1 \leq m \leq n)$$

or the empty string. The *concatenation* of strings $S_1, \dots, S_k$ is the string $S_1 S_2 \dots S_k$, which consists of the symbols of S_1 followed by those of S_2, and so on.

Claim. Any nonempty terminal segment of a P-wff is a concatenation of P-wff.

Proof of Claim. Let A be a P-wff. We prove that any nonempty terminal segment of A is a concatenation of P-wff, by induction on the construction of A. If $A = p$ this is obvious. If $A = \neg B$, then any nonempty terminal segment of A is either A itself or a nonempty terminal segment of B, so it is a concatenation of P-wff, by induction hypothesis. Finally, if $A = *BC$, then any terminal segment of A is either A itself or the concatenation of a terminal segment of B with C or else a terminal segment of C, so again, by induction hypothesis, we are done.

Example. Consider the indicated terminal segment of the given P-wff:

$$\Rightarrow\Rightarrow \underbrace{s\neg t\neg \vee p \wedge \neg q s}$$

It is the concatenation of the following P-wff:

$$\underbrace{s}\ \underbrace{\neg t}\ \underbrace{\neg \vee p \wedge \neg q s}$$

We can now complete the proof of the lemma (and hence the theorem):

Suppose A is a P-wff and B is a P-wff which is a proper initial segment of A, so that $A = BC$, with C a nonempty terminal segment of A. Then we have

$$w(A) = w(B) + w(C).$$

But $w(A) = w(B) = 1$ and, as $C = C_1 \dots C_n$ for some P-wwf $C_1, \dots, C_n$,

$$w(C) = \sum_{i=1}^{n} w(C_i) \geq 1,$$

which is a contradiction. $\dashv$

Remark. From the preceding, it follows that any nonempty terminal segment of a P-wff is the concatenation of a *unique* sequence of P-wff.