Logic is to language and meaning as mathematics is to physical science. In order to understand how sentences – which are what compose language – work, it is necessary to learn to find their logical structure. This doesn't tell us everything about the sentence, but it **does** provide a skeleton on which the other parts of the sentence can be fleshed out, the same way $E=MC^2$ doesn't tell you **how** matter converts into energy, just that it does.

Luckily, logic is much, much simpler to learn than other varieties of mathematics. For one thing, numbers and arithmetic play no role whatsoever in logic. Logic is all about relationships between chunks of meaning, which are called ***Propositions***. Propositions are composed of ***Predicates*** and ***Arguments***. Roughly speaking, **Propositions** are the skeletons of **sentences**; **Predicates** are the skeletons of **verbs** (or **adjectives**), and **Arguments** are the skeletons of **nouns** (or **pronouns**). Almost everything else in a sentence is pretty much ignored in logic; it's only interested in the skeletal parts.

The way you use logic in grammar is to take a sentence and dissect it into its logical skeleton. For instance, *Bill ran* is a very simple sentence, with only one verb and one noun. It corresponds to the Proposition RUN (BILL). (We use SMALL CAPS to represent logical formulae, to distinguish them from actual chunks of language, which are *italicized*.) Notice that the verb in the sentence *Bill ran* is in the past tense, but the Predicate in RUN (BILL) is not; logic normally ignores tense, so this is also the skeleton of *Bill runs*.

Also notice that the Predicate RUN comes first, followed by the Argument BILL, in parentheses, even though the words *Bill* and *ran* occur in the opposite order in the sentence. This is **always** the order in a Proposition, no matter where words occur in the actual sentence; the Predicate comes first, followed by its Argument (or Arguments – there can be up to three, separated by commas), in parentheses.

Let's try another sentence: *Bill kicked the ball*. The corresponding Proposition is KICK (BILL, BALL). By convention, the first Argument inside the parentheses is the ***Subject*** of the sentence (in English, all sentences must have a Subject, and it's prototypically the ***Agent*** of the action denoted by the verb); the next Argument, if there is one, is the ***Direct Object.*** A sentence whose corresponding Proposition has <u>two</u> Arguments is called a ***Transitive*** sentence; a sentence with only <u>one</u> Argument (like *Bill ran*) is called ***Intransitive***.

Notice, once again, that tense is ignored in the Proposition; notice also that the definite article *the* is also ignored. If the Object had been plural *balls* instead of singular *ball*, **that** would have been ignored, too. Logic is only concerned with skeletons and the connections of their parts.

Some Predicates can take <u>three</u> Arguments; these are called ***Bitransitive***. Most bitransitive Predicates have to do with **transfer** or **movement** of something from one place (or person) to another. For example, *Bill threw the ball to John* is bitransitive, and its corresponding Proposition is THROW (BILL, BALL, JOHN). The three Arguments, in order, are Subject, Direct Object, and ***Indirect Object***; they have, respectively, the roles (meanings) of ***Source***, ***Trajector***, and ***Goal***. Semantically, the Trajector is what moves, and it moves from Source to Goal, usually with energy provided by the Source. Notice that the preposition *to* is ignored here; the Arguments will appear in this order by convention.

And that's **it**. That's how Propositions are formed: ***PREDICATE (ARGUMENTS)***. This is called ***Predicate Calculus***. We will return to Predicate Calculus shortly.

The other part of logic has to do with combinations and elaborations of Propositions, and it's called ***Propositional Calculus***. It's much simpler than Predicate Calculus.

One elaboration of a Proposition is called ***negation*** and consists of using *not*, as in the sentence: *Bill didn't kick the ball*. The corresponding Proposition is ¬ KICK (BILL, BALL). The symbol '¬' is a logical ***truth functor***; it's pronounced simply as 'not'. It can be applied to any Proposition, yielding a new Proposition, with opposite ***Truth Value***.

One of the simplest kinds of combination is a functor called ***conjunction*** and consists of putting *and* between two sentences: *Bill kicked the ball and John caught it*. The corresponding Proposition is KICK (BILL, BALL) ∧ CATCH (JOHN, BALL). The symbol '∧' is the logical *and* truth functor; it is defined in logical terms, and is not to be confused with the English conjunction *and*, though it's pronounced 'and'. It can connect any two Propositions, yielding a new Proposition. Note that the pronoun *it* in this sentence refers to *the ball*, so BALL is the Argument used for *it* in the Proposition; logic has no pronouns, only Arguments. If we were being very, very careful and formal, we'd have to mark both instances of the argument BALL with the same ***referential index*** '*i*' as BALL_i, just to point out that they did in fact refer to the **same** ball.

Another, related, kind of combination is a functor called ***alternation*** and consists of putting *or* between two sentences: *Bill kicked the ball or John kicked the ball*. The corresponding Proposition is KICK (BILL, BALL) ∨ KICK (JOHN, BALL). The symbol '∨' is the logical *or* truth functor; like ∧, it is defined in logical terms and is not at all identical to the English conjunction *or,* but it can combine any two Propositions and yield a new Proposition. Note that this sentence can be shortened to *Bill or John kicked the ball*; the logical statement is the same, either way.

¬, ∧ and ∨ are defined, like all truth functors, in means of ***Truth Tables***. Logic is so simple because it abstracts away from what we think of as meaning and largely ignores it; in logic, there are only **two** possible 'meanings' (also called Truth Values) for **any** Proposition: ***True*** and ***False***. Therefore it is easy to specify precisely the truth values for any combined Proposition, since there is a very limited number of possible combinations. In particular, if ***p*** and ***q*** represent **any** Proposition at all, the Truth Tables for ¬, ∧ and ∨ are:

¬	*p*	*p*	∧	*q*	*p*	∨	*q*
F	T	T	T	T	T	T	T
		T	F	F	T	T	F
T	F	F	F	T	F	T	T
		F	F	F	F	F	F

The first two columns say that if **p** is True, then **¬p** is False, and vice versa. The truth values of the Proposition **¬p** in each line are conventionally put under the functor (here '¬') in a Truth Table. That's all there is for negation, since it only applies to a single Proposition (i.e, it's a ***monadic functor***), and that Proposition has only two possibilities. We say there are only two lines in the Truth Table of ¬.

By contrast, ∧ and ∨ are ***dyadic functors***, combining **two** Propositions each. This means there are 4 possible combinations of T and F for the two Propositions, so there are four lines in their Truth Tables. The top line (where both p and q are True), and the bottom line (where both are False), are identical for ∧ and ∨. In the first case, both ∧ and ∨ are True, while in the last case, they're both False. It is the middle two lines where the difference lies: if **either** p or q is **False**, then p∧q is False; but if **either** p or q is **True**, then p∨q is True. The result is that the Truth Table of ∧ can be summarized (just reading down the values of the column under ∧) as **TFFF**, while the Truth Table of ∨ is **TTTF**.

In fact, this is actually pretty close to our ordinary understanding of what *and* and *or* mean: in order for *p and q* to be True, **both** Propositions have to be True, whereas in order for *p or q* to be True, only **one** needs to be True. (One might note that there is a different use of *or* in English, which represents a different dyadic functor, with the Truth Table **FTTF**, i.e, *p or q* can be False when **both** p **and** q are True; we usually emphasize this sense in English by using *either…or*, as in *Either John or Bill kicked the ball*. This functor is called **exclusive *or*** (or ***disjunction***) and it is important in computer design, where it is symbolized ***XOR***. Logic normally does not distinguish between these two functors, and neither will we, unless necessary.)

Since truth functors are defined exclusively in terms of Truth Tables, which amount to permutations of T and F, it follows that there are 4 monadic truth functors (2^2=4): **TT, TF, FF**, and Negation, which is **FT**; except for Negation, the others are all irrelevant. Likewise, there are 16 possible dyadic truth functors (2^4=16), ranging from **TTTT** through **FFFF**. Most of these are merely mathematical curiosities, but there are two more dyadic functors besides ∧ and ∨ that are of interest in natural language. Below are the Truth Tables for them:

p	≡	***q***	***p***	⊃	***q***
T	**T**	**T**	**T**	**T**	**T**
T	**F**	**F**	**T**	**F**	**F**
F	**F**	**T**	**F**	**T**	**T**
F	**T**	**F**	**F**	**T**	**F**

The first one is ***logical equivalence***, symbolized by '≡'. The Truth Table for ≡ is **TFFT**; i.e, p ≡ q is True whenever p and q are either both True **or** both False. Notice that, under this definition, any Proposition that is always True, like *Two plus three equals five*, is logically equivalent to **all** other Propositions that are always True, like *December 31 is the last day of the year*, or *The moon is not made of green cheese*. While this is far from what we ordinarily mean by 'equivalent', remember that the only meanings allowed in logic are **T** and **F**; this is a logical result.

The second one is ***material implication***, symbolized by '⊃' (called 'horseshoe'). Unlike all the other dyadic functors we've seen so far, implication is not ***symmetric***; i.e, while it doesn't matter what order we write p ≡ q, p ∧ q, or p ∨ q, it **does** matter when we write p ⊃ q. Put logically, p ∧ q is equivalent to q ∧ p [in symbols, **(p ∧ q) ≡ (q ∧ p)**], but p ⊃ q is **not** equivalent to q ⊃ p [in symbols, **¬ ((p ⊃ q) ≡ (q ⊃ p))**]. (Note, parenthetically, how the parentheses pile up fast in logical formulae; we'll return to this a bit later.) The Truth Table for ⊃ is **TFTT**.

⊃ is often glossed as 'if…then' and is the basis of logical accounts of deductive thought, especially syllogisms. The first line of the Truth Table for ⊃ supports this: p ⊃ q is True if both p and q are True. Certainly that's what we mean by saying things like *If Caesar is dead, then he can't come to tea*. We say this is a ***valid inference***, and deductively correct; that's what the **T** in the Truth Table means. The most direct mode of the syllogism, called ***Modus Ponens***, goes like this: p ⊃ q is True; but p is True; therefore q is True [in symbols, ((p ⊃ q) ∧ p) ⊃ q]. From the correctness of the statement above, and the fact that Caesar is dead, we conclude that indeed he can't come to tea. (Logic is not rocket science)

The second line of the Truth Table says that p ⊃ q is False if p is True but q is False; i.e, if the premise is True but the conclusion is False, the inference is **invalid**. This also tends to support the 'if…then' interpretation of ⊃: we would like to be able to say that *If two plus two equals four, then the Moon is made of green cheese* is **not** a valid inference. So far, so good.

It is the last two lines of the Truth Table for ⊃ that give people the most trouble. They seem to say that *If two plus two equals three, then the Moon is made of green cheese* is True, **and** so is *If two plus two equals three, then the Moon is* ***not*** *made of green cheese.* Certainly this is not what anybody means by 'if…then', and it can hardly be said to be good clear thinking. Or can it?

As it turns out, these **are** necessary for correct reasoning. Consider the last line, for instance: the mode of the syllogism called ***Modus Tollens*** goes like this: p ⊃ q is True; but q is False; therefore p is False [in symbols, ((p ⊃ q) ∧ ¬q) ⊃ ¬p); an example is *If he was hungry, he would have eaten. But he didn't eat. Therefore he wasn't hungry.*]. In order for Modus Tollens to work, p ⊃ q has to be True (i.e, a correct inference) when both p and q are False.

Taken together, the last two lines of the Truth Table for ⊃ say that from a False premise, **any** Proposition at all can be correctly deduced; in other words, be **very** careful not to assume False premises. This is what lies at the root of Proof by Contradiction, which starts by assuming some Proposition (p) and then showing that it leads to a contradiction: (p ⊃ q) ∧ (p ⊃ ¬q). The conclusion of this contradiction must then be that p is False (¬p). In other words, the last two lines of the Truth Table for ⊃ allow us to reason from contradictions, and are thus very valuable, if confusing.

Truth Tables can also be used to prove statements that are formed with truth functors. For instance, a theorem of logic called *Reduction of Implication* shows that ⊃ is not in fact necessary, since it's equivalent to a combination of ¬ and ∨. The statement of the theorem is: ((p ⊃ q) ≡ (¬p ∨ q)): in words, "P implies Q is equivalent to Not-P or Q". The truth table has only four lines, since there are only two arguments; here it is with only the values for ***p*** and ***q*** filled in – the ***basic state*** of a Truth Table:

(*p*	⊃	*q*)	≡	(¬	*p*	∨	*q*)
T	**T**	**T**			**T**		**T**
T	**F**	**F**			**T**		**F**
F	**T**	**T**			**F**		**T**
F	**T**	**F**			**F**		**F**

The basic state simply distinguishes the four possible combinations of the truth values of *p* and *q*.and thus the four lines of the table; we have added the canonical **TFTT** truth table for ⊃, since it's already been given. So far we don't know what the truth value of the proposition as a whole is; but there are simple rules for finding out. As in algebra, we work from the inside out. Here the 'inside' is the simply negated proposition ¬*p*. So we fill in the truth values for that, which go in the column under the functor ¬.

(*p*	⊃	*q*)	≡	(¬	*p*	∨	*q*)
T	**T**	**T**		**F**	**T**		**T**
T	**F**	**F**		**F**	**T**		**F**
F	**T**	**T**		**T**	**F**		**T**
F	**T**	**F**		**T**	**F**		**F**

Now we can fill in the truth values for the *or*, using the values for ¬*p* instead of *p* and *or*-ing them with the values for *q* to remove the parentheses:

(*p*	⊃	*q*)	≡	(¬	*p*	∨	*q*)
T	**T**	**T**		**F**	**T**	**T**	**T**
T	**F**	**F**		**F**	**T**	**F**	**F**
F	**T**	**T**		**T**	**F**	**T**	**T**
F	**T**	**F**		**T**	**F**	**T**	**F**

Now we have removed all the parentheses; the second column contains the truth values of the left-hand proposition, and the seventh column now contains the truth values of the right-hand proposition. Note that these two columns are identical: **TFTT**. That means that whenever one is true the other is true, and whenever one is false the other is false. That's a definition of logical equivalence; as the final truth table shows, the truth value of the entire theorem, shown in the fourth column, is **TTTT**.

(p	⊃	***q)***	≡	**(¬**	***p***	∨	***q)***
T	**T**	**T**	**T**	**F**	**T**	**T**	**T**
T	**F**	**F**	**T**	**F**	**T**	**F**	**F**
F	**T**	**T**	**T**	**T**	**F**	**T**	**T**
F	**T**	**F**	**T**	**T**	**F**	**T**	**F**

That is, this proposition is true under **all** circumstances, true no matter what its constituent propositions are, true no matter whether **they're** true or false, true of everything, everywhere, and forever. It's ***logically true***. It's been ***proven***. It's a ***theorem***.

It also means that we may substitute '¬p ∨ q' for 'p ⊃ q' whenever we like in any proposition, without changing the truth value (*salva veritate*). There are thousands (actually, a countably infinite number, but we don't have time for all of them) of other theorems that are or can be proven. Proving them is the business of logic, for the most part.

However, since we are not logicians, we have a much simpler task. We need only know (and be able to apply) the truth tables of ¬, ∨, ∧, ⊃, and ≡. In addition, we need to be aware of the following important theorems, and their meaning (some of these are actually ***axioms*** or ***postulates***, instead of theorems; the difference only matters to logicians, since all are logically true). As an exercise, you should test your understanding of each by putting it into reasonably clear English.

- *Law of Contradiction*: ¬(p ∧ ¬p)
- *Law of Excluded Middle*: p ∨ ¬p
- *Double Negative Law*: ¬(¬p) ≡ p
- *Reduction of Equivalence*: (p ≡ q) ≡ ((p ⊃ q) ∧ (q ⊃ p))
- *Reduction of Implication*: (p ⊃ q) ≡ (¬p ∨ q)
- *Disjunctive Syllogism*: ((p ∨ q) ∧¬q) ⊃ p
- *Hypothetical Syllogism*: ((p ⊃ q) ∧ (q ⊃ r)) ⊃ (p ⊃ r)
- *Modus Ponens*: ((p ⊃ q) ∧ p) ⊃ q
- *Modus Tollens*: ((p ⊃ q) ∧ ¬q) ⊃ ¬p
- *Contrapositive Law*: (p ⊃ q) ≡ (¬q ⊃ ¬p)
- *De Morgan's Laws*: ¬(p ∨ q) ≡ (¬p ∧ ¬q)
  ¬(p ∧ q) ≡ (¬p ∨ ¬q)

Now that we have mastered Propositional Calculus (or at least all of it that one needs to know) we can return to Predicate Calculus. While Propositional Calculus is concerned with Propositions (and sentences) as a whole, and therefore studies rules of logic that apply to any Proposition, Predicate Calculus is concerned with the **parts** of Propositions (and sentences): ***Predicates*** and ***Arguments***.

Recall from p.1 that there are three main kinds of Propositions (and sentences) in terms of how many arguments occur with them: the ***Intransitive***, or ***One-place*** predicates, with only one Argument; the ***Transitive***, or ***Two-place*** predicates, with two Arguments; and the ***Bitransitive***, or ***Three-place*** predicates, with three Arguments.
(To this we may add the ***Impersonal***, or ***Zero-place*** predicates, with no Arguments at all, as in *It's raining*, represented logically as RAIN (); this is a special and rather rare case and doesn't feature into logic much, though it's important in English syntax, since English requires every sentence to have a subject, whether or not it represents a Proposition with an Argument.)

[By the way, at this point in the discussion I intend to stop capitalizing the basic terms used in logic and linguistics, on the argument that by now they should be familiar to you. I **will** continue to write new terms in ***bold italics***, and occasionally to ***Capitalize*** them, when they first appear in the text. That is your cue to learn to recognize them when they pop up in the discussion again. Clear? OK, onward.]

Every proposition **must** have a predicate and (except in the case of impersonals) at least one argument, in just the same way every sentence of English (or any other language) must have a verb or an adjective or a noun that ***predicates*** some idea. (Note that '*to predicate*', the verb in bold italics in the preceding sentence, while obviously related to the noun *predicate*, is pronounced differently; the final syllable in the **verb** *predicate* /'prɛdəˌket/ has a secondary stress and is pronounced like *Kate*, while the final syllable in the **noun** *predicate* /'prɛdəkət/ is unstressed and is pronounced like *cut*. The noun *predication* /ˌprɛdə'keʃən/ is derived from the verb and means 'the act of predicating.')

And just what does 'to predicate' **mean**? Well, it's the basic abstraction in a sentence: it's what one says **about** the arguments. So, in *John is tired*, what one predicates of *John* is the state of **being tired**: TIRED (JOHN); *tired* is an adjective, so *be tired* is a ***predicate adjective.*** In *Bill is a doctor*, what one predicates of Bill is the quality of **being a doctor**: DOCTOR (BILL): *doctor* is a noun, so *be a doctor* is a ***predicate noun***. In *John kicked the ball*, what one predicates of *John* and *ball* is the action of **kicking**: KICK (JOHN, BALL); *kick* is a verb, and verbs are the ***prototype*** predicates, so we don't have to call *kick* a 'predicate verb'.

Note that predicate adjectives and predicate nouns, not being prototype predicates, are required by English grammar to have some form of *be* appearing right before them, and that predicate nouns in English require in addition an indefinite article *a* following the *be*. This is a general and quite automatic fact about English grammar, however, and it is ignored by logic since it doesn't contribute to meaning; thus we say that TIRED, DOCTOR, and KICK are the predicates of these propositions, and basically ignore the automatic *be* and *a* in the predicates, the same way we ignored the automatic *the* in *the ball*.

Predication is an **extremely** general concept; it is at the basis of all human language, and is the first significant linguistic concept acquired by most children when they learn their native language. At the 'two-word' stage in 'bottom-up' language acquisition,

when children learn to put together more than one word (eg. *Doggie bye-bye*), what they are in fact learning is predication. What is being predicated of the argument *doggie* in *Doggie bye-bye* is some concept of departure or absence, derived from the behavior label *bye-bye*, and used as a predicate in this proposition. It is an immensely impressive intellectual achievement to take a term that is learned as a vocal accompaniment of a hand gesture and generalize it to a predicate of departure, capable of applying to any argument.

In addition, predication is ***recursive***. That is, one can predicate something of a proposition as a whole, not just a noun. Put another way, propositions can be arguments of other propositions, in much the same way that mathematical functions can be 'composed with' other functions: y = f (g(x)). For example, in *Bill wants Mary to leave*, there are two propositions (or ***clauses***), each with its own predicate and its own arguments. One of the propositions (or *clauses*) is ***dependent***, or ***subordinate***, to the other proposition, which is the ***independent***, or ***main***, or ***matrix*** clause. The subordinate clause is *(for) Mary to leave*, and its corresponding proposition is LEAVE (MARY). The main clause **contains** the subordinate clause as an argument (the direct object – what Bill wants – of its predicate *want*); the corresponding proposition is WANT (BILL, LEAVE (MARY)). Note the extra parentheses.

[By the way, there are many other kinds of subordinate clauses; this is just one example. To be precise, this is a ***direct object infinitive complement*** clause, a kind of ***noun clause***.]

This kind of recursion can be carried on indefinitely, leading to arbitrarily long sentences: *Bill said that Mary thought that John intended Bill to say that Mary thought that John intended Bill to say that* ...[SAY (BILL, THINK (MARY, INTEND (JOHN, (SAY (BILL, THINK (MARY, INTEND (JOHN, SAY (BILL, …)))))))))]. It is very, very common; the majority of English sentences contain one or more subordinate clauses, of which ***complement*** clauses like these are the most important, logically and syntactically.

Arguments, too, can be complicated. Arguments are normally nouns or pronouns, referring to ***entities***, in the same way that predicates refer of ***events*** or ***states***. However, while logic normally ignores articles like *a* and *the* that modify nouns, and normally treats adjectives as predicates even when they modify nouns, there is one additional kind of little word attached to arguments that is crucial to the meaning of sentences and figures prominently in logic. That little chunk of meaning is the ***quantifier***.

It has been known since Aristotle that there were two different kinds of proposition that had different effects in syllogisms. Suppose one considers the difference between *All Hoosiers are right-handed* and *Some Hoosiers are right-handed*. If it is the case that **all** Hoosiers are right-handed, then it follows trivially that **some** of them are. However, the reverse is not true: from the fact that **some** Hoosiers are right-handed, we cannot conclude anything about **all** Hoosiers. The difference is clearly due to the words *all* and *some*, which are examples of quantifiers, a ***syntactic category*** ('Part of Speech') in all languages, and also a form of logical functor (actually, in formal logic, quantifiers are called ***operators***, not functors; once again, the difference is only relevant to a logician). There are many different kinds of quantifier in natural language, but logic uses only two abstract varieties: the ***universal quantifier*** (*each, every, any, all*; symbolized by ∀) and the ***existential quantifier*** (*some, there exists, at least one*; symbolized by ∃).

In formulas, these quantifier symbols are placed at the beginning of the proposition, before the predicate, which in turn comes before the arguments that the quantifiers modify. In order to match up each quantifier with the argument that it ***binds***, one uses ***variable*** symbols (typically *x* and *y*, just like algebra). Thus *Somebody ate peaches* is symbolized as (∃*x*) EAT (*x*, PEACHES); that is, in some case, there was a person that ate peaches. Likewise, *Everybody ate peaches* is symbolized as (∀*x*) EAT (*x*, PEACHES); that is, in every case, there was a person that ate peaches.

One can be more specific about the universe of discourse of quantifiers; in particular, it is often useful to use ***restricted quantifiers*** with limited reference. For instance, in the examples of the preceding paragraph, it is taken for granted that the argument *x* that is bound by the quantifiers is restricted to ('quantifies over') **people**. However, this is not a given in logic, and ought to be specified if one is being precise. One way of doing this is to include the restriction in the quantifier statement. Thus instead of the formal statements in the preceding paragraphs, one could say (∃*x*: PERSON (*x*)) EAT (*x*, PEACHES) and (∀*x*: PERSON (*x*)) EAT (*x*, PEACHES) ; that is, in either some or all cases **of persons**, the person ate peaches. If everyone is clear about the presuppositions, this is not necessary, but one should always be prepared to be more specific if necessary, since that is what logic is for.

We say that a quantifier ***binds*** an argument it refers to, or that it ***quantifies over*** cases of such an argument. It is possible to bind more than one argument in a proposition (though each binding requires a different quantifier), and in such a case, we find that the formal notation provides a benefit, disambiguating a potentially ambiguous sentence. Thus, in the sentence *Every boy read some book*, there are two possible readings: either there is some book (say *Moby Dick*) that every boy read, or each boy read a unique book (which might possibly be the same in any two cases, but need not be – Bill read *Moby Dick*, Frank read *Small Gods* and so did Joe, but Mark read *Tropic of Capricorn*).

These two cases are represented by the handy logical fact that if there are two quantifiers in any proposition, they can occur in two possible orders, since they are not part of the arguments they bind. Thus the two cases above are represented as (∃*y*: BOOK (*y*)) (∀*x*: BOY (*x*)) READ (*x*, *y*), and (∀*x*: BOY (*x*)) (∃*y*: BOOK (*y*)) READ (*x*, *y*). That is (respectively), there is some book such that for every boy, that boy read the book; and, for every boy, there is some book such that that boy read the book.

We say, in the first case, that the existential quantifier ∃ is ***outside the scope*** of the universal quantifier ∀, while the universal is ***inside the scope*** of the existential; or, alternatively, that the existential quantifier has ***wide scope***, while the universal has ***narrow scope***. Reverse 'universal' and 'existential' to describe the second case; either way, the ***scope*** of a quantifier is the relevant technical term. This kind of ambiguity comes about because natural language normally requires quantifiers to modify (and thus appear with) the arguments they bind, which does not always allow scope to be specified. This is called a ***quantifier-crossing*** (or Q-Crossing) ***ambiguity*** in the syntactic trade; in logic, of course, it does not arise because the scope of quantifiers **must** be specified.

Quantifiers may only bind arguments, not predicates. This restriction results in a theory called ***First-Order Quantified Predicate Calculus***. The 'first-order' part means

that quantification is restricted to arguments; if quantification over predicates were allowed, it would be Second Order. The significance here is that first-order quantified predicate calculus has been mathematically proven to be ***consistent***, and thus useful for all kinds of mathematical and engineering purposes, while second-order has been proven to be ***inconsistent***, which means that one can't rely on logical calculi to produce correct results with it. Thus logicians (and mathematicians and engineers) are restricted to first-order calculi. This doesn't mean that human language is, of course; just that logic can only go so far in modelling it.

Similar kinds of ambiguities arise when negatives occur with quantifiers: *All the boys didn't leave* can mean **either** that some left but some didn't, **or** that they all stayed. Respectively, these correspond to ¬(∀*x*: BOY (*x*)) LEAVE (*x*) or (∀*x*: BOY (*x*)) ¬LEAVE (*x*).

That is, it makes a difference where a negative appears with respect to a quantifier. A negative on one side of a quantifier has a different meaning from one on the other side. This ***quantifier-negative*** (or Q-Neg) ***ambiguity*** is in fact a consequence of De Morgan's Laws, which we saw before in terms of the functors ∧ and ∨. In fact, the quantifers ∀ and ∃ have a natural relation to the functors ∧ and ∨ (respectively).

(∃*x*) refers to **some** *x*: i.e, x_1 **or** x_2 **or** x_3 **or** x_4 **or** …; while (∀*x*) refers to **all** *x*: i.e, x_1 **and** x_2 **and** x_3 **and** x_4 **and** … Thus De Morgan's Laws, stated in terms of quantifiers instead of functors, where *P* represents **any** predicate at all, are: ¬(∀*x*) P (*x*) ≡ (∃*x*) ¬P (*x*) and ¬(∃*x*) P (*x*) ≡ (∀*x*) ¬P (*x*). That is, if P is not true **for all** *x*, then **there is some** *x* such that P is false; while if **there is no** *x* such that P is true for that *x*, then **for all** *x*, P is false.

Indeed, these are only two of the forms in which De Morgan's Laws (and all the rest of logic) appear. Logic is the basis of all mathematics, including ***set theory***, and also of electronic ***circuit design***. The accompanying illustration shows two more ways to state De Morgan's Laws in these fields. In addition, it shows one further logical variation, called ***modal logic***, which is extremely useful in analyzing natural language.

Modality (or ***mode***, or ***mood***) is a logical or semantic notion that appears, like negation and quantification, in every human language. Indeed, modality is intrinsically linked with negation and quantification, and has many complex interactions with them. The basic ideas behind modality are the concepts of ***possibility*** and ***necessity***. Any word in a sentence that deals with these concepts in any way can be considered a ***modal term*** of some sort, and modal logic may be applied to propositions that contain it. In English, the set of modal terms (or just ***modals***) includes the ***modal auxiliary verbs*** *must, can, could, may, might, shall, should, will, would,* and sometimes *need* and *dare*. It also includes adjectives like *(im)possible, (im)probable,* and *(un)likely*, and adverbs like *maybe, possibly, probably,* and *perhaps*.

Possibility and necessity are inverses of one another in exactly the same way that ∃ and ∀ are, and that ∨ and ∧ are. If some proposition is **necessarily true**, then it is **not possible** for it to be false; likewise, if it is **possibly true**, then it is **not necessary** for it to be false. Modal logic uses the symbol □ (square) for **necessary**, and the symbol ◊ (diamond) for **possible**. Thus, for any proposition *p*, □p means 'p is necessarily true' or just 'p is necessary', while ◊p means 'p is possibly true' or just 'p is possible'.

De Morgan's Laws in modal logic use the reciprocal definitions of □ and ◊: ◊p ≡ ¬□¬p, and □p ≡ ¬◊¬p; or (respectively) if p is **possible**, then it **not necessarily false**, and if p is **necessary**, then it is **not possibly false**. Possibility, looked at this way, means that p is true in **some** set of circumstances, while necessity means that p is true in **all** sets of circumstances. In other words, in the correct context, □ is equivalent to ∀ (and thus ∧), while ◊ is equivalent to ∃ (and thus ∨). So □ and ◊ commute under negation in exactly the same way ∀ and ∃ do, which is exactly the same way ∧ and ∨ do.

¬□p ≡ ◊¬p	¬(∀*x*) P (*x*) ≡ (∃*x*) ¬P (*x*)	¬(p ∧ q) ≡ (¬p ∨ ¬q)
¬◊p ≡ □¬p	¬(∃*x*) P (*x*) ≡ (∀*x*) ¬P (*x*)	¬(p ∨ q) ≡ (¬p ∧¬q)

Recall that in many sentences the corresponding propositions have a large number of parentheses. For instance, a rather ordinary sentence like *Bill wants to begin to learn to speak Spanish* has the awkward associated proposition WANT (BILL, BEGIN (BILL, LEARN (BILL, SPEAK (BILL, SPANISH)))). This kind of detail makes a notation system too cumbersome to be of any use; if one is always having to match parentheses, there is no attention left to the novel conclusions that a calculus is supposed to produce. There are two different ways to solve this problem. Logic takes one and linguistics the other.

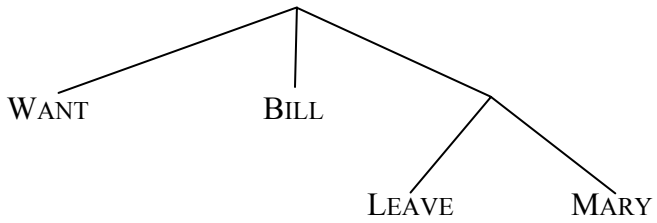
The logical solution is to do away with parentheses entirely, replacing them with a ***prefixal*** notation. Thus, instead of putting the functor ∨ **between** p and q, as p∨q, one might put it **before** them, as ∨pq. This is often called ***Polish notation*** or ***PN***, since it was originated by the Polish logican Jan Łukasiewicz (1878-1956), and most English speakers can neither spell nor pronounce his name (/wu'kaz̨evitc̨/). A variant is ***reverse Polish notation*** or ***RPN***, often used in programming languages (or Hewlett-Packard calculators), in which the functors (or in HP calculators, the arithmetic operation) is suffixal, following the arguments, as pq∨. PN and RPN are completely equivalent, and either may easily be translated into the other. PN is still the most common logical variety.

In PN, each functor or operator is given a different letter (always upper-case, occasionally Greek) and its arguments are expressed in lower-case letters. PN formulas are read left-to-right, and no expression is complete until **all** its arguments are **completely** expressed. Thus no parentheses are needed. In Łukasiewicz notation, ¬p is **Np** (*N* for *Negativ* – Łukasiewicz was writing in German); p∧q is **Kpq** (*K* for *Konjunction*); p∨q is **Apq** (*A* for *Alternation*); p⊃q is **Cpq** (*C* is a mnemonic for ⊃); p≡q is **Epq**; ◊p is **Mp** (*M* for *Möglich*); □p is **Lp** (*L* for *Logisch*); (∀*x*) is **Px**; (∃*x*) is **Sx.** Some equivalences are shown on the accompanying illustration of De Morgan's Laws. Łukasiewicz notation works exactly the same as 'standard' notation (which is, roughly speaking, the notation used by Whitehead and Russell in their seminal work *Principia Mathematica*) in terms of truth tables and other logical niceties, but – since it does not require parentheses – it is much more machine-washable (since it is easily interpretable using only a simple pushdown stack) and finds much more favor in computational circles. For our purposes it is necessary only to be aware of it and be able if necessary to translate it into standard notation.

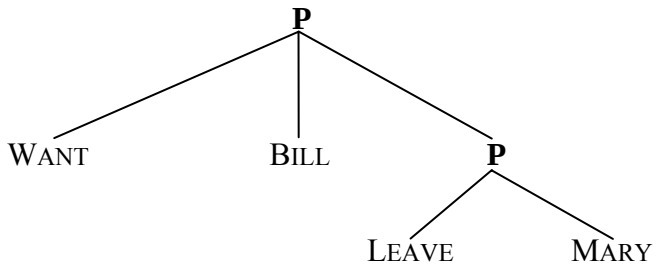
That is the logical solution to the problem of parentheses: change the notation to make the constituents easier to push around. The linguistic solution, on the other hand, is more concerned with making the data more easily accessible than with making the notation simpler, and thus it crucially takes note of the fact that the problem of parentheses

arises from trying to make everything appear on one line. Parentheses are nothing more than a way to represent two dimensions in one. That is, when we use multiple parentheses in WANT (BILL, LEAVE (MARY)), we are noting that the proposition LEAVE (MARY) is contained in (subordinate to, dependent on) the entire proposition (main clause, matrix sentence).

If we treat such predicational subordination as another dimension (say, the **vertical** dimension, which is after all the metaphor suggested by terms like *dependent* and *subordinate*), then we might get rid of the parentheses in WANT (BILL, LEAVE (MARY)) (at the cost of using more paper) by writing it in two dimensions, something like:



This mode of representation is called a ***tree diagram***, though clearly this is metaphorical also, since the 'trees' are upside down. Each node may have a label, producing a ***labelled tree***, which looks like this:



where **P** designates **P**roposition. We could label the **Pr**edicate and **A**rgument nodes as well, but the convention of putting the predicate to the left makes that unnecessary; in this case, for instance, it is clear that the lower, dependent, subordinate **P** serves as the second argument of the predicate WANT in the higher P; the order is the same as the order in the linear proposition WANT (BILL, LEAVE (MARY)). However, the parentheses and commas that are required to indicate the relations between the parts of the one-dimensional proposition are not needed in the two-dimensional diagram. In a proposition as simple as this, the difference between the two is small, but as the structures grow more complex, it is much easier to see the relationships among the parts of the structure.

Since propositions are logical structures modelling sentences (or clauses), it is common to use *S* instead of *P* to label propositions in such logical tree structures in linguistics. Either way, of course, the ***constituent structure*** of the proposition is made much more clear through such tree diagrams, and parentheses may be dispensed with. Tree diagrams are used, and much more highly elaborated, in grammatical analyses, but they are all based on the simple kind of predicate-and-argument diagram shown here.

There are many more logical details one could talk about, but mastering these is generally enough for anyone interested in English grammar. Attached are some diagrams and exercises covering the concepts discussed here.

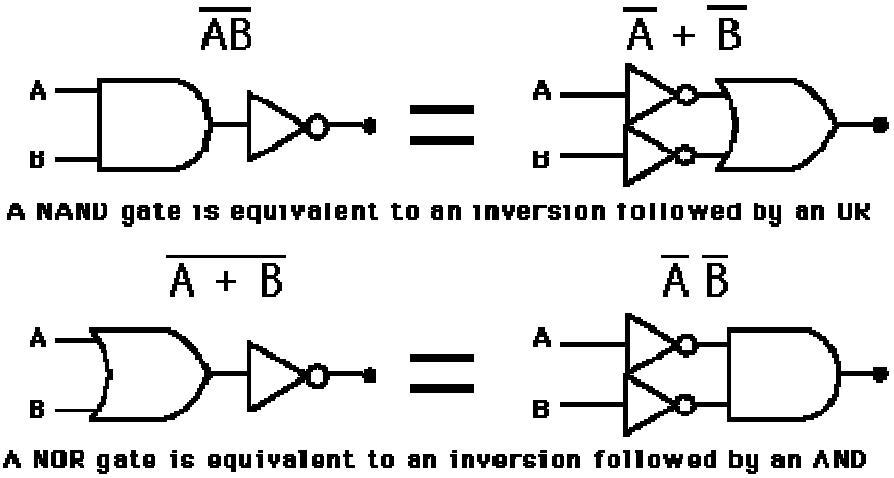
De Morgan's Laws, as stated in various formal theories and notations

1. Set Theory

The Complement of the Intersection of any number of sets is Equivalent to the Union of their Complements. $\overline{\bigcap_j A_j} \equiv \bigcup_j \overline{A_j}$

The Complement of the Union of any number of sets is Equivalent to the Intersection of their Complements. $\overline{\bigcup_j A_j} \equiv \bigcap_j \overline{A_j}$

2. Circuit Diagrams



3. Propositional Calculus

(Not (p And q)) is Equivalent to *((Not p) Or (Not q))*

$\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$	'Standard Notation' (*Principia Mathematica*)
ENKpqANpNq	'Polish Notation' (Łukasiewicz)

(Not (p Or q)) is Equivalent to *((Not p) And (Not q))*

$\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$	*PM*
ENApqKNpNq	Łuk

4. Modal Logic

(p is Not Necessary) is Equivalent to *(Not p is Possible)*

$\neg\Box p \equiv \Diamond\neg p$	*PM*
ENLpMNp	Łuk

(p is Not Possible) is Equivalent to *(Not p is Necessary)*

$\neg\Diamond p \equiv \Box\neg p$	*PM*
ENMpLNp	Łuk

5. First-Order Quantified Predicate Calculus

(Not (For Every x, φ(x))) is Equivalent to *(For Some x, Not φ(x))*

$\neg(\forall x)\ \phi(x) \equiv (\exists x)\ \neg\phi(x)$	*PM*
ENPxϕxSxNϕx	Łuk

(Not (For Some x, φ(x))) is Equivalent to *(For Every x, Not φ(x))*

$\neg(\exists x)\ \phi(x) \equiv \forall(x)\ \neg\phi(x)$	*PM*
ENSxϕxPxNϕx	Łuk

I.	Grammatical Sentence	Logical Proposition
1.	Bill sings.	SING (BILL)
2.	Bill enjoys singing.	ENJOY (BILL, SING (BILL))
3.	Mary enjoys Bill's singing.	ENJOY (MARY, SING (BILL))
4.	Bill sings well.	GOOD (SING (BILL))
5.	Mary sees Bill.	SEE (MARY, BILL)
6.	Mary saw Bill.	SEE (MARY, BILL) *or* PAST (SEE (MARY, BILL))
7.	Bill is tired.	TIRED (BILL)
8.	Bill was tired.	TIRED (BILL) *or* PAST (TIRED (BILL))
9.	Bill sang a song.	SING (BILL, SONG) *or* ...
10.	Mary thinks that Bill is tired.	THINK (MARY, TIRED (BILL))
11.	Mary wants to leave.	WANT (MARY, LEAVE (MARY))
12.	Mary wants Bill to leave.	WANT (MARY, LEAVE (BILL))
13.	Mary said that Bill was tired.	SAY (MARY, TIRED (BILL))
14.	Mary told Bill that she was tired.	TELL (MARY, BILL, TIRED (MARY))
15.	Mary told Bill that he was tired.	TELL (MARY, BILL, TIRED (BILL))
16.	Mary told Bill to leave.	TELL (MARY, BILL, LEAVE (BILL))
17.	Mary thought that the man was tired.	PAST (THINK (MARY, TIRED (MAN)))
18.	Mary thinks that Bill left.	THINK (MARY, PAST (LEAVE (BILL)))
19.	Mary said that Bill had left.	PAST (SAY (MARY, PAST (LEAVE (BILL)))
20.	Mary told Frank that Bill had left.	PAST (TELL (MARY, FRANK, PAST (LEAVE (BILL)))

Above are twenty English sentences with their corresponding propositions expressed in Predicate Calculus. Note the PREDICATE (ARGUMENTS) notation.

Find or create twenty more sentences of approximately equal complexity and give their corresponding propositions. You will have to determine which terms are predicates, and which other terms are their arguments. Note that not all logical arguments are expressed – compare (2) and (3), for instance.

Since logic ignores articles, as well as person, number, and tense, you may do so also whenever it is not relevant to the exposition. For example, in (17-20) tense **is** relevant, and is therefore indicated, but since there is no basic propositional difference between (7) and (8), tense can be profitably ignored.

Hints: (a) Don't forget to balance parentheses.
(b) Avoid quantifiers like *all* or *some*; they appear in Exercise 2.

II. Express in both 'Standard' and 'Polish' notation the following propositions. For each, special peculiarities and instructions are appended.

1. *All the boys don't speak Spanish.*
   NB: Since this is ambiguous, it will need two expressions, one for each sense.

2. *Bill may not like logic, but he can't ignore it.*
   NB: Treat *but* as equivalent to And, and both *may* and *can* as Possible.

3. *If you build it, they will come.*
   NB: Ignore tense.

4. *Nobody likes a wise guy.*
   NB: Treat *a wise guy* as an existential quantifier,
   e.g, 'Some *x* such that WISE-GUY (*x*)';
   be careful with the placement of the quantifiers and the negative.

5. *You can fool all of the people some of the time,*
   and some of the people all of the time.
   NB: Treat *fool* as a 3-place predicate with a time argument,
   e.g, FOOL (YOU, *x*, *t*) means 'You fool *x* at time *t*'.

6. *'Oh, life is a glorious cycle of song,*
   A medley of extemporanea;
   And love is a thing that can never go wrong,
   And I am Marie of Roumania.'
   — Dorothy Parker (1893-1967)
   NB: Treat the predicates as single items (e.g, MARIE-OF-ROUMANIA (I)), and adjacent propositions without conjunctions as if connected by And. Comment on the truth values of the individual conjuncts.

7. *If that answer's correct, then I'm a monkey's uncle.*
   NB: Treat the predicates as single items, and comment on the truth value of the proposition as a whole. How does the Contrapositive Law apply here?

8. *He has to be either here or there, but he's not here.*
   NB: Treat *but* as And, and comment on the conclusion to be drawn.

9. *Though everybody likes Raymond, not everybody likes his mother.*
   NB: Treat *though* as And. Is order of conjoined propositions significant?

10. *Buy today and get 0% interest.*
    NB: Expand this sentence by supplying missing arguments before logicizing it; some of them are indefinite, to be supplied by context, and therefore not a problem for logic. Be careful of the *and*: it doesn't really represent logical And, but rather a different functor. Which one? Is this an imperative?