Propositional logic :

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

Examples :

“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) ¬

and (conjunction) ∧

or (disjunction) ∨

implies (implication) ⇒

iff (equivalence) ⇔

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. ¬) 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” (¬) 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 “or” (∨) is true whenever either or both

of its components are true. That is, ∨ is the so-called “inclusive or”.

The domain or universe or universe of discourse for a predicate variable is the set of values that may be

assigned to the variable.

set of P(x) is the set of all elements t of U such that P(t) is true, ie

{t ∈ U|P(t) is true}

Example: U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

P(x): “x is even”.

The truth set is: {2, 4, 6, 8, 10}

Predicate logic :

The Universal Quantifier: ∀

Turn predicates into propositions by assigning values to all variables:

Predicate P(x): “x is even”

Proposition P(6): “6 is even”

The other way to turn a predicate into a proposition: add a quantifier like “all” or “some” that indicates the number of values for

which the predicate is true.

The symbol ∀ is called the universal quantifier.

The universal quantification of P(x) is the statement “P(x) for

all values x in the universe”, which is written in logical notation as:

∀xP(x) or sometimes ∀x ∈ D, P(x).

Ways to read ∀xP(x):

For every x, P(x)

For every x, P(x) is true

For all x, P(x)

The Existential Quantifier: ∃

Definition: The symbol ∃ is call the existential quantifier and

represents the phrase “there exists” or “for some”. The existential

quantification of P(x) is the statement “P(x) for some values x in

the universe”, or equivalently, “There exists a value for x such that

P(x) is true”, which is written ∃xP(x).

Note: If P(x) is true for at least one element in the domain, then

∃xP(x) is true. Otherwise it is false.

Note: Let P(x) be a predicate and c ∈ U (U = domain).

The following implications are true:

∀xP(x) → P(c)

P(c) → ∃xP(x)

Example: ∃x [x is prime] where U = Z

FOPL

Propositional vs. Predicate Logic

•In propositional logic, each possible atomic fact requires a

separate unique propositional symbol.

•If there are n people and m locations, representing the fact

that some person moved from one location to another

requires nm2

separate symbols.

•Predicate logic includes a richer ontology:

-objects (terms)

-properties (unary predicates on terms)

-relations (n-ary predicates on terms)

-functions (mappings from terms to other terms)

•Allows more flexible and compact representation of

knowledge

Move(x, y, z) for person x moved from location y to z.

Sentence → AtomicSentence

| Sentence Connective Sentence

| Quantifier Variable Sentence

| ¬Sentence

| (Sentence)

AtomicSentence → Predicate(Term, Term, ...)

| Term=Term

Term → Function(Term,Term,...)

| Constant

| Variable

Connective → ∨ | ∧ | ⇒ | ⇔

Quanitfier → ∃ | ∀

Constant → A | John | Car1

Variable → x | y | z |...

Predicate → Brother | Owns | ...

Function → father-of | plus | …

Conversion of FOPL into clauses :

Steps to convert a sentence to clause form:

Eliminate all <=> connectives by replacing each instance of the form (P <=> Q) by the equivalent expression ((P => Q) ^ (Q => P))

Eliminate all => connectives by replacing each instance of the form (P => Q) by (~P v Q)

Reduce the scope of each negation symbol to a single predicate by applying equivalences such as converting ~~P to P; ~(P v Q) to ~P ^ ~Q; ~(P ^ Q) to ~P v ~Q; ~(Ax)P to (Ex)~P, and ~(Ex)P to (Ax)~P

Standardize variables: rename all variables so that each quantifier has its own unique variable name. For example, convert (Ax)P(x) to (Ay)P(y) if there is another place where variable x is already used.

Eliminate existential quantification by introducing Skolem functions. For example, convert (Ex)P(x) to P(c) where c is a brand new constant symbol that is not used in any other sentence. c is called a Skolem constant. More generally, if the existential quantifier is within the scope of a universal quantified variable, then introduce a Skolem function that depend on the universally quantified variable. For example, (Ax)(Ey)P(x,y) is converted to (Ax)P(x, f(x)). f is called a Skolem function, and must be a brand new function name that does not occur in any other sentence in the entire KB.

Example: (Ax)(Ey)loves(x,y) is converted to (Ax)loves(x,f(x)) where in this case f(x) specifies the person that x loves. (If we knew that everyone loved their mother, then f could stand for the mother-of function.

Remove universal quantification symbols by first moving them all to the left end and making the scope of each the entire sentence, and then just dropping the "prefix" part. For example, convert (Ax)P(x) to P(x).

Distribute "and" over "or" to get a conjunction of disjunctions called conjunctive normal form. Convert (P ^ Q) v R to (P v R) ^ (Q v R), and convert (P v Q) v R to (P v Q v R).

Split each conjunct into a separate clause, which is just a disjunction ("or") of negated and un-negated predicates, called literals.

Standardize variables apart again so that each clause contains variable names that do not occur in any other clause.

Example

Convert the sentence (Ax)(P(x) => ((Ay)(P(y) => P(f(x,y))) ^ ~(Ay)(Q(x,y) => P(y))))

Eliminate <=>

Nothing to do here.

Eliminate =>

(Ax)(~P(x) v ((Ay)(~P(y) v P(f(x,y))) ^ ~(Ay)(~Q(x,y) v P(y))))

Reduce scope of negation

(Ax)(~P(x) v ((Ay)(~P(y) v P(f(x,y))) ^ (Ey)(Q(x,y) ^ ~P(y))))

Standardize variables

(Ax)(~P(x) v ((Ay)(~P(y) v P(f(x,y))) ^ (Ez)(Q(x,z) ^ ~P(z))))

Eliminate existential quantification

(Ax)(~P(x) v ((Ay)(~P(y) v P(f(x,y))) ^ (Q(x,g(x)) ^ ~P(g(x)))))

Drop universal quantification symbols

(~P(x) v ((~P(y) v P(f(x,y))) ^ (Q(x,g(x)) ^ ~P(g(x)))))

Convert to conjunction of disjunctions

(~P(x) v ~P(y) v P(f(x,y))) ^ (~P(x) v Q(x,g(x))) ^ (~P(x) v ~P(g(x)))

Create separate clauses

~P(x) v ~P(y) v P(f(x,y))

~P(x) v Q(x,g(x))

~P(x) v ~P(g(x))

Standardize variables

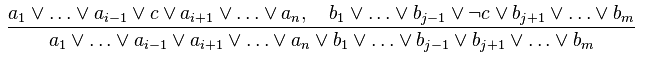
~P(x) v ~P(y) v P(f(x,y))

~P(z) v Q(z,g(z))

~P(w) v ~P(g(w))

Resolution :

The resolution rule in propositional logic is a single valid inference rule that produces a new clause implied by two clauses containing complementary literals. A literal is a propositional variable or the negation of a propositional variable. Two literals are said to be complements if one is the negation of the other (in the following, \lnot c is taken to be the complement to c). The resulting clause contains all the literals that do not have complements. Formally:



where

all as, bs and c are literals,

the dividing line stands for entails

The clause produced by the resolution rule is called the resolvent of the two input clauses. It is the principle of consensus applied to clauses rather than terms.[3]

When the two clauses contain more than one pair of complementary literals, the resolution rule can be applied (independently) for each such pair; however, the result is always a tautology.

Modus ponens can be seen as a special case of resolution of a one-literal clause and a two-literal clause.