9 Background: Predicate Logic

9.1Introduction

Variable assignments were developed in a branch of logic called **predicate logic**. As this is a notational system widely used in semantics, a short introduction will be included here.

In Statement Logic (cf. section 4) the basic expressions were statements that could be true or false, and that could form complex statements using the connectives ¬ , , , → and ↔ .

Predicate Logic also has ways to refer to expressions that make up simple statements. One particularly simple and important version, called **first-order predicate logic**, has the following types of expressions:

· **Terms**, which refer to entities in the universe of discourse;

· **Predicates**, which refer to relations between entities in the universe of discourse.

Terms come in two varieties, called **names** or **individual constants**, and **variables**. Variables are comparable to pronouns or empty elements in natural language.

Predicates are given as relations. For example, a two-place predicate that represents the English verb *love* would be rendered as a relation symbol L that takes a pair of names like s, m and forms a sentence L(s, m) (pronounced "L of s, m", which is meant to be interpreted as "s loves m"). This contrasts with the notation we have employed so far, which could be rendered as L(m)(s). Recall that this notation is better suited for linguistic purposes, as it reflects the fact that the object argument is syntactically closer to the verb than the subject argument.

A predicate that takes one term argument is called "one-place". We have **n-place predicates**, for any natural number n. Of course, in natural language we only find predicates with relatively few arguments, but there is no problem in defining, say, a 17-place predicate. We also have 0-place predicates; these are predicates that need no argument at all to form a sentence, that is, 0-place predicates are basic sentences.

Sentences are also called **formulas**.

The only two quantifiers of predicate logic are the **universal** quantifier and the **existential** quantifier, written " " and " ", followed by a variable. Typical quantified statements have the following form:

(1) a. x L(s,x) 'There is an x such that L of s, x', 'There is an x such that s loves x'
b. x L(s,x) 'For all x it holds that L of s, x', 'For all x it holds that s loves x'

The glosses show that the quantifiers of predicate logic are to be understood like English *something* or *everything*.

Some other quantifiers can be expressed indirectly in first-order predicate logic, for example the negative quantifier:

(2) ¬ x L(s, x) 'It is not the case that there is an x such that s loves x',
= 'There is no x such that s loves x'

9.1.1 Syntax of Predicate Logic

A more formal definition of the language of first-order predicate logic follows. We start with the basic expressions of each category:

(3) a. Terms:
i) We have **variables** x_0, x_1, x_2,... (I will use x, y, z for x_0, x_1, x_2).
ii) We have **individual constants** c_0, c_1, c_2, ...(I will use small letters as abbreviations, e.g. m for c_0).

b. Predicates:
We have, for each natural number n, **n-place predicate constants** $P^n{}_0$, $P^n{}_1$, $P^n{}_2$, $P^n{}_3$, etc. (I will use capital letters for abbreviation, e.g. S for $P^1{}_0$, L for $P^2{}_0$, etc.)

Rules for forming well-formed expressions of each category:

(4) a. If α is a basic expression of a category, then α is a well-formed expression of that category.
b. If π is an n-place predicate and τ_1, ... τ_n are terms, then $\pi(\tau_1,...\tau_n)$ is a formula.
c. If Φ, Ψ are formulas, then ¬Φ, [Φ Ψ], [Φ Ψ], [Φ → Ψ], [Φ ↔ Ψ] are formulas.
d. If Φ is a formula and v is a variable, then vΦ, vΦ are formulas.

An example of well-formed formulas of predicate logic:

(5) $x_0[P^1{}_0(x_0) \to x_1P^2{}_0(x_0, x_1)]$
(with the abbreviations introduced above, we may write x[S(x) → yL(x, y)])

9.2Quantifiers and their variables

There are various notions that apply to quantifiers and variables.

In a formula like vΦ, we say that Φ is the **scope** of the quantifier occurrence v, and similarly for vΦ. For example:

a) xS(x) Scope of x: S(x)

b) xS(x) P(x) Scope of x: S(x)

c) x[S(x) P(x)] Scope of x: [S(x) P(x)]

d) x[S(x) → yL(x,y)] Scope of x: [S(x) → yL(x,y)] Scope of y: L(x,y)

e) x y[S(x) → L(x,y)] Scope of x: y[S(x) → L(x,y)] Scope of y: [S(x) → L(x,y)]

An occurrence v′ of a variable v is called **bound** if v′ is in the scope of a quantifier v or v. If an occurrence of a variable is not bound, it is called **free**. Examples:

f) All the variable occurrences in the above examples (a-e)are bound, for except x in P(x) in case (b).

g) S(x) Occurrence of x is free.

h) x S(y) Occurrence of y is free.

i) x L(x,y) Occurrence of y is free.

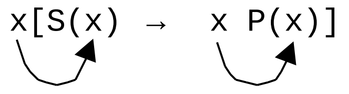
j) [x yL(x,y) S(x)] The second occurrence of x is free.

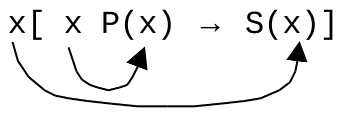
An occurrence v′ of a variable v is **bound by a quantifier occurrence** Q′ of v (or v) iff

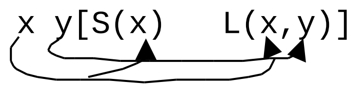
- v′ is in the scope of the quantifier occurrence Q′;
- there is no occurrence Q″ of a quantifier v or v such that Q″ is in the scope of Q′, and v′ is in the scope of Q″ (that is, Q′ is the "closest" qantifier occurrence that binds variables v, in terms of scope).

Examples:

k) In examples (a)-(e), the variable occurrences are bound by their respective quantifiers.

l) 
x[S(x) → x P(x)] Binding relations as indicated.

m) 
x[x P(x) → S(x)] Binding relations as indicated.

n) 
x y[S(x) L(x,y)] Binding relations as indicated.

The syntactic rule (4) allows for **vacuous quantification**. That is, an expression like xS(m) is a well-formed formula, although x binds no variable. This is harmless when it comes to semantic interpretation, as we will see.

A formula that does contain variable occurrences that are not bound by any quantifier is called **open**. A formula that is not open is **closed**. (Sometimes, we call a closed formula **sentence** or **statement**).

9.3The Expressiveness of Predicate Logic

Predicate Logic is a highly expressive formal language, that is, it can be used to formulate many statements. This is the reason why it is widely used in mathematics and philosophy. There is even one proposal of an artificial human language that is partly based on predicate logic, a language called *Loglan* by James Cooke Brown.

Before we go into the semantics of expressions of predicate logic, let us see with a few examples what can be expressed in predicate logic.

So far, we have introduced quantifiers that correspond to English *everyone* and *someone* (or *everything* and *something*, depending on whether we are speaking about persons or things). But in many cases, quantification is explicitly **restricted** to certain entities. For example, *every girl* expresses a universal quantification restricted to girls, and *some boy* expresses an existential quantification restricted to boys. Do we need special quantifiers for noun phrases like *every girl* or *some boy*? The answer is No: We can use the general universal and existential quantifier to express sentences containing these noun phrases. For example:

(6) *Some girl sang*: x[G(x) S(x)]
("something is a girl and sang",
"there is at least one value for x for which [G(x) S(x)] is true")

(7) *Every girl sang*: $x[G(x) \rightarrow S(x)]$
(“everything is such that if it is a girl, then it sang”,
“for every value of x it holds that $[G(x) \rightarrow S(x)]$ is true.

Note that the last formula happens to be true even if there is no girl. This is due to the logic of the conditional, $\rightarrow$: If G(x) is false, then $G(x) \rightarrow S(x)$ will be true. It is unclear whether the English sentence *every girl sang* is true when there is no girl (consider the dialogue: *Have you invested every case of malaria? - Yes, there was none.*) But it certainly is not false -- it is, perhaps, inappropriate. The predicate logic formula does not capture that.

Aristotle assumed four forms of general statements (which he called “particular affirmative”, “universal affirmative”, “universal negative”, and “particular negative”), and they all can easily be captured with our quantifiers together with the sentential operators negation, conjunction and conditional:

(8) a. particular affirmative, e.g. *some girl sang*: $x[G(x) \quad S(x)]$
b. universal affirmative, e.g. *every girl sang*: $x[G(x) \rightarrow S(x)]$
c. universal negative, e.g. *no girl sang*: $\neg \quad x[G(x) \quad S(x)]$
d. particular negative, e.g. *some girl didn’t sing*: $\neg \quad x[G(x) \rightarrow S(x)]$

As for the representation of particular negative statements, notice that *some girl didn’t sing* could also be expressed by *not every girl sang.*

Predicate logic is much more flexible than Aristotle's logic. For example, we can treat sentences with more than one quantifier, like *Everyone likes someone*. Some more examples of that type:

(9) a. *Every girl likes some boy*: $x[G(x) \rightarrow \quad y[B(y) \quad L(x,y)]]$
b. *Some girl likes every boy*: $x[G(x) \qquad y[B(y) \rightarrow L(x,y)]]$
c. *Some girl likes every boy that likes her*:
$x[G(x) \qquad y[[B(y) \quad L(y,x)] \rightarrow L(x,y)]]$
d. *Every girl likes some boy that likes some girl*:
$x[G(x) \rightarrow \quad y[B(y) \qquad z[G(z) \quad L(y,z)] \quad L(x,y)]]$

Assume that we have an identity relation =. This is just a special two-place relation, where we have that for any two terms $\alpha, \beta, \alpha=\beta$ is true if the meaning of α and the meaning of β is the same. Then we can render even more complicated sentences:

(10)a. *Every girl likes two (or more) boys.*
$x[G(x) \rightarrow \quad y \ z[B(y) \quad B(z) \quad \neg y=z \quad L(x,y) \quad L(x,z)]]$
b. *Every girl likes exactly one boy.*
$x[G(x) \rightarrow \quad y[B(y) \quad L(x,y) \qquad z[[B(z) \quad L(x,z)] \rightarrow y=z]]]]$
c. *Every girl likes a different boy.*
$x[G(x) \rightarrow \quad y[B(y) \quad L(x,y) \quad \neg \quad z[G(z) \quad \neg z=x \quad L(z,y)]]]$
d. *Every girl and every boy like each other.*
$x \quad y[[G(x) \quad B(y)] \rightarrow [L(x,y) \quad L(y,x)]]$

This great flexibility and expressiveness makes predicate logic an ideal tool for expressing mathematical statements. It also often allows us to clarify readings of natural-language expressions. But we will see that predicate logic has its limits, and that natural langu, while being less precise, is actually more expressive than predicate logic.

9.4 The Semantics of Predicate Logic

As in the case of statement logic, predicate logic can only tell us what the meaning of a complex expression is, given the meaning of the basic expressions. The only difference is that we have more categories of basic expressions, and more syntactic rules. For the interpretation of predicate logic we also introduce the notion of a **model**. A model specifies, in essence, how basic expressions (constants and variables) should be interpreted. In particular, a model for predicate logic consists of two things:

(11)a. A set D, the **domain of discourse** (which we have called D_e).
  b. A function F that gives us the **meaning of constant basic expressions**.

A model M is given as a pair of a domain and a meaning assignment, D, F .

The meaning assignment for constant basic expressions F will depend on the specific assumptions we have for the meanings of basic expressions. However, F must assign meanings of the proper type. In particular,

- individual constants are mapped to elements of D,
- basic formulas (0-place predicates) are mapped to elements of {0,1},
- basic 1-place predicates will be mapped to subsets of D,
- basic 2-place predicates will be mapped to two-place relations in D,
- in general, basic n-place predicates will be mapped to n-place relations in D

Example of a model for a simple predicate logic with names m, s, j, one-place predicates G, B, S, and two-place predicate L:

(12)a. M = D, F ,
  b. D = {Mary, Sue, John}
  c. F(m) = Mary, F(s) = Sue, F(j) = John,
   F(G) = {Mary, Sue}, F(B) = {John}, F(S) = {Mary}
   F(L) = { Mary, John , Sue, Mary , John, John , Mary, Sue }

F will give us just the meaning of the basic constants. The meaning of the variables is given by **variable assignments**. These are functions from the set of variables to the domain D. In contrast to the variable assignments we have used in the last section, variable assignments in predicate logic are not partial but **total**, that is, they map each variable to some value. I will use g, g′ etc. for variable assignments. An example of a variable assignment for the model given above (only the first four values are specified):

(13)g: VAR $\rightarrow$ D, where g(x) = Mary, g(y) = John, g(z) = Mary, g(x3) = Sue, ...
  that is, g = { x_0, Mary , x_1, John , x_2, Mary , x_3, Sue , …}

The meaning of **complex expressions** will be given in terms of the meanings of their immediate parts, as follows:

Let M = D, F be a model for predicate logic that satisfies the requirements given above, and let g be a variable assignment. Then we can specify the meaning of expressions α with respect to M and g, commonly written as $[\![\alpha]\!]^{M,g}$, in the following way:

(14)a. If α is a basic constant expression, then $⟦α⟧^{M,g}$ = F(α).

b. If α is a variable, then $⟦α⟧^{M,g}$ = g(α).
(That is, constants are interpreted by F, variables are interpreted by g).

c. If If π is an n-place predicate and $τ_1$, ... $τ_n$ are terms, then
$⟦π(τ_1,...τ_n)⟧^{M,g}$ = 1 if $⟦τ_1⟧^{M,g},...⟦τ_n⟧^{M,g} \quad ⟦π⟧^{M,g}$, else 0.

d. If Φ, Ψ are formulas, then

$⟦¬Φ⟧^{M,g} = 1$ if $⟦Φ⟧^{M,g} = 0$, else 1.
$⟦[Φ \quad Ψ]⟧^{M,g} = 1$ if $⟦Φ⟧^{M,g} = 1$ and $⟦Ψ⟧^{M,g} = 1$, else 0.
$⟦[Φ \quad Ψ]⟧^{M,g} = 0$ if $⟦Φ⟧^{M,g} = 0$ and $⟦Ψ⟧ = 0$, else 1.
$⟦[Φ → Ψ]⟧^{M,g} = 0$ if $⟦Φ⟧^{M,g} = 1$ and $⟦Ψ⟧^{M,g} = 0$, else 1.
$⟦[Φ ↔ Ψ]⟧^{M,g} = 1$ if $⟦Φ⟧^{M,g} = ⟦Ψ⟧^{M,g}$, else 0.

These are the usual interpretation rules of the operators of statement logic.

e. The rule for identity "=", which is a special two-place relation:
$⟦α=β⟧^{M,g}$ = 1 if $⟦α⟧^{M,g} = ⟦β⟧^{M,g}$, else 0.

(Note that the first occurrence of "=" is a symbol of the object language, the second is part of the definition scheme (we could have written "$=_{def}$"), and the third is s a symbol of the metalanguage).

f. If Φ is a formula and x is a variable, then
$⟦\quad xΦ⟧^{M,g}$ = 1 iff for some element d, d $\quad$ D, it holds that $⟦Φ⟧^{M,g[x/d]}$ = 1.
$⟦\quad xΦ⟧^{M,g}$ = 1 iff for every element d, d $\quad$ D, it holds that $⟦Φ⟧^{M,g[x/d]}$ = 1.

Here, g[x/d] stands for **assignment variants** (cf. section 8); g[x/d] is the assignment that is like g except that it maps the variable x to the object d.

Let us consider some examples, all with respect to the model given above, (12), and the variable assignment

(15) g = { x, John , y, Sue , z, Mary , x_3, Sue ...}.

We start with some formulas that do not involve quantifiers.

(16) $⟦G(m)⟧^{M,g}$ = 1 if $⟦m⟧^{M,g} \quad ⟦G⟧^{M,g}$,
if F(m) $\quad$ F(G), which is the case, hence $⟦G(m)⟧^{M,g}$ = 1.

(17) $⟦[G(m) \quad L(m,j)]⟧^{M,g}$ = 1 if $⟦G(m)⟧^{M,g}$ = 1 and $⟦L(m,j)⟧^{M,g}$ =1
if $⟦m⟧^{M,g} \quad ⟦G⟧^{M,g}$ and $⟦m⟧^{M,g}, ⟦j⟧^{M,g} \quad ⟦L⟧^{M,g}$
if F(m) $\quad$ F(G) and F(m), F(j) $\quad$ F(L)
This is the case, hence: 1.

(18) $⟦G(x)⟧^{M,g}$ = 1 if $⟦x⟧^{M,g} \quad ⟦G⟧^{M,g}$,
if g(x) $\quad ⟦G⟧^{M,g}$, which is not the case, hence: 0.

Now a few formulas with quantifiers:

(19) $⟦\quad xG(x)⟧^{M,g}$ = 1 iff for **some** element d, d $\quad$ D, it holds that $⟦G(x)⟧^{M,g[x/d]}$ = 1.

We have a choice of three cases, d = Mary, d = John, d = Sue.

Take d = Mary,
we have $\llbracket G(x)\rrbracket^{M,g[x/Mary]} = 1$ iff
$\llbracket g\rrbracket^{M,g[x/Mary]} \quad \llbracket G\rrbracket^{M,g[x/Mary],}$ iff
g[x/Mary](x)   F(G), iff
Mary   {Mary, Sue}, which is true, according to our model.

Hence we have found a d as required, hence $\llbracket \ \ xG(x)\rrbracket^{M,g} = 1$.

(20) $\llbracket \ \ x[G(x) \rightarrow S(x)]\rrbracket^{M,g} = 1$
iff for **every** element d, d   D, it holds that $\llbracket [G(x) \rightarrow S(x)]\rrbracket^{M,g[x/d]} = 1$.

We have three elements to consider: d = Mary, d = John, d = Sue.

Case d = Mary:
$\llbracket [G(x) \rightarrow S(x)]\rrbracket^{M,g[x/Mary]} = 0$, iff
$\llbracket G(x)\rrbracket^{M,g[x/Mary]} = 1$ and $\llbracket S(x)\rrbracket^{M,g[x/Mary]} = 0$, iff
$\llbracket x\rrbracket^{M,g[x/Mary]} \quad \llbracket G\rrbracket^{M,g[x/Mary]}$ and $\llbracket x\rrbracket^{M,g[x/Mary]} \quad \llbracket S\rrbracket^{M,g[x/Mary]}$, iff
g[x/Mary](x)   F(G) and g[x/Mary](x)   F(S), iff
Mary   {Mary, Sue} and Mary   {Mary}, which is
True and False, respectively, in our model.

Hence we know that $\llbracket [G(x) \rightarrow S(x)]\rrbracket^{M,g[x/Mary]} \neq 0$, hence = 1.

Case d = John:
$\llbracket [G(x) \rightarrow S(x)]\rrbracket^{M,g[x/John]} = 0$, iff
(similar derivation as above)
John   {Mary, Sue} and John   {Mary}
False   True

Hence we know that $\llbracket [G(x) \rightarrow S(x)]\rrbracket^{M,g[x/John]} \neq 0$, hence = 1.

Case d = Sue:
$\llbracket [G(x) \rightarrow S(x)]\rrbracket^{M,g[x/Sue]} = 0$, iff
(similar derivation as above)
Sue   {Mary, Sue} and Sue   {Mary}
True   True

Hence we know that $\llbracket [G(x) \rightarrow S(x)]\rrbracket^{M,g[x/Sue]} = 0$.

So we were **not** able to show $\llbracket [G(x) \rightarrow S(x)]\rrbracket^{M,g[x/d]} = 1$ for all d   D.

Hence we know that $\llbracket \ \ x[G(x) \rightarrow S(x)]\rrbracket^{M,g} = 0$.

The formula " x[G(x) → S(x)]" shows a way how predicate logic can mimick restricted quantification. If G stands for *girl* and S for *Sleep*, then this is a good representation of the English sentence *Every girl sleeps*. But, technically, it say something quite different, namely: for every x it holds: if x is a girl, then x sleeps. Notice that this is a quantification over *all* entities.

Let us finally consider an example with more than one quantifier:

(21) ⟦ x yL(x,y)]⟧^{M,g} = 1 iff
for every d, d D, ⟦ yL(x,y)⟧$^{M,g[x/d]}$ = 1

We have three cases to consider: d = Mary, d = Sue, d = John.

Case d = Mary:
⟦ yL(x,y)⟧$^{M,g[x/Mary]}$ = 1 iff there is a d, d D, such that ⟦L(x,y)⟧$^{M,g[x/Mary][y/d]}$ = 1.

We have a choice of three cases, d = Mary, d = John, d = Sue.

Take d = John.
⟦L(x,y)⟧$^{M,g[x/Mary][y/d]}$ = 1, as
⟦x⟧$^{M,g[x/Mary][y/John]}$, ⟦y⟧$^{M,g[x/Mary][y/John]}$ ⟦L⟧$^{M,g[x/Mary][y/John]}$, as
g[x/Mary][y/John](x), g[x/Mary][y/John](y) F(L), as
Mary, John F(L), as
Mary, John { Mary, John , Sue, Mary , John, John , Mary, Sue },
which is True.

Case d = Sue:
⟦ yL(x,y)⟧$^{M,g[x/Sue]}$ = 1 iff there is a d, d D, such that ⟦L(x,y)⟧$^{M,g[x/Sue][y/d]}$ = 1.

We have a choice of three cases, d = Mary, d = John, d = Sue.

Take d = Mary. We find: ⟦L(x,y)⟧$^{M,g[x/Sue][y/d]}$ = 1.

Case d = John,
⟦ yL(x,y)⟧$^{M,g[x/John]}$ = 1 iff there is a d, d D, such that ⟦L(x,y)⟧$^{M,g[x/John][y/d]}$ = 1.
We have a choice of three cases, d = Mary, d = John, d = Sue.
Take d = John. We find: ⟦L(x,y)⟧$^{M,g[x/John][y/John]}$ = 1.

Hence: ⟦ x yL(x,y)]⟧^{M,g} = 1.

And now an example in which the order of the quantifiers is different:

(22) ⟦ x yL(x,y)⟧^{M,g} = 1 iff
there is a d, d D, such that ⟦ yL(x,y)⟧$^{M,g[x/d]}$ = 1,
iff for every d′, d′ D, ⟦L(x,y)⟧$^{M,g[x/d][y/d']}$ = 1.

(I use here a different meta-language variable d′ to keep the two places distinct, which facilitates the discussion)

We have a choice of three cases: d = Mary, d = Sue, d = John.

Take d = Mary.
Is it the case that for every d′, d′ D, ⟦L(x,y)⟧$^{M,g[x/d][y/d']}$ = 1 ?
No; one counterexample is d′ = Mary.

Take d = Sue.
Is it the case that for every d′, d′ D, ⟦L(x,y)⟧$^{M,g[x/d][y/d']}$ = 1 ?
No; one counterexample is d′ = John.

Take d = John.

Is it the case that for every d', d' D, $[\![L(x,y)]\!]^{M,g[x/d][y/d']} = 1$?

No again; one counterexample is d' = Mary.

Hence we were not able to find a d, as required.

Hence $[\![$ x yL(x,y)$]\!]^{M,g} = 0$.

We see that the order of quantifiers can make a semantic difference. It may be that with respect to the same model, the formula x yL(x,y) is true, but the formula y xL(x,y) is false.

9.5The Importance of Predicate Logic

The flexibility of predicate logic is quite evident. No wonder that it became a favorite tool for describing mathematical structures, formalizing theories, etc. In addition to that, powerful algorithms have been proposed that allow us to find out whether one predicate logic statement follows from another statement (so-called **proof** or **deduction systems**). We will not go into these issues here (see for a discussion Partee, ter Meulen & Wall ch. 7). Here just some important facts about predicate logic:

· A predicate logic that just allows for quantification over individuals is called **first-order predicate logic**. If the logic also allows quantification over sets (predicate meanings), we call it **second order**. An example of a second-order expression is X[X(a) X(b)] (roughly, there is a property that a and b share).

· For first-order predicate logic there are deduction systems that are **complete**. That is, whenever a sentence entails another sentence, the deduction system will show that this is the case.
Of course, the deduction systems are also **sound**. That is, whenever the deduction system tells us that from one statement Φ we can derive another statement Ψ, then it is guaranteed that Φ entails Ψ.
If we use "Φ Ψ" for "Φ entails Ψ", a semantic notion, and "Φ |— Ψ" for "the deduction system allows us derive Ψ from Φ", then this property of first-order predicate logic can be expressed as follows:

(23)There are deduction systems for first-order predicate logic for which it holds for any two sentences Φ, Ψ: [Φ Ψ] if, and only if, [Φ |— Ψ].
(This is the famous completeness theorem for first-order predicate logic, due to Kurt Gödel, 1930).

· For second-order predicate logic, it has been shown that there cannot be a complete deduction system (this is the even more famous incompleteness theorem due to Gödel, 1931, together with his completeness theorem perhaps the most important discovery ever made in logic).

· In order to formalize mathematics (even simple arithmetics), we need second-order predicate logic. Hence there is no complete deduction system for mathematics. Or, there are true sentences in mathematics that cannot be proved by any deduction system.

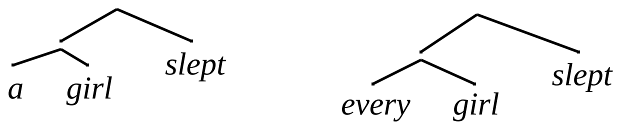
· We will see that, for the semantics of natural language, the complexity of first-order predicate logic is not sufficient either.

9.6A Comparison with Generalized Quantifier Theory

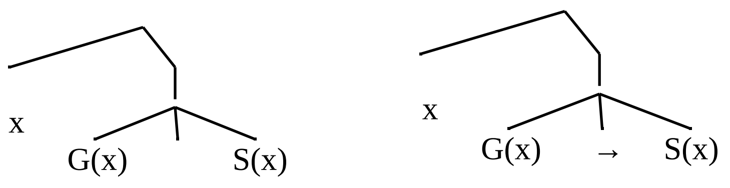
First-order predicate logic was mainly developed as a tool to express statements in mathematics and other fields in a precise, unambiguous way. The question arises how it compares to quantification in natural languages. Of course, natural languages are highly ambiguous (they have no brackets, they have a limited set of distinctive pronouns etc.). But they are also more expressive than first-order predicate logic. And it turns out that Generalized Quantifier Theory is a better model for quantification in natural language than first-order predicate logic.

For one thing, Generalized Quantifier Theory allows for a better fit between syntactic structure and semantic intepretation. Consider the way how the following sentences are expressed in fist-order predicate logic and in GQ theory:

(24)a. English:

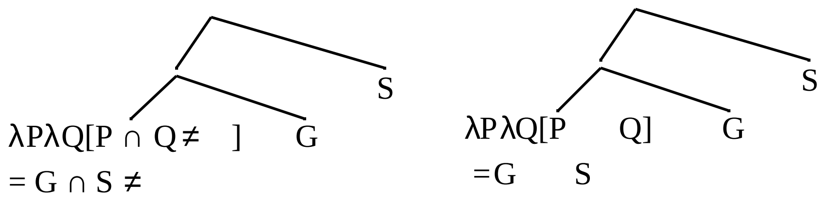


b. Predicate Logic:



:

c. Generalized Quantifiers:



Notice that an indefinite NP like *a girl* is expressed by an existential quantifier and a sentence connective, a conjunction. A universal NP is expressed by a universal quantifier and a conditional. There is hardly any correspondence between the structure of the English sentences and the structure of the formulas in predicate logic. But there is a perfect correspondence between the English sentences and the GQ analyses.

Also, Generalized Quantifiers are more expressive. True, some statements with generalized quantifiers can be expressed in predicate logic, though it's often awkward:

(25)a. No girl slept.
λPλQ[P ∩ Q =](G)(S)
¬ x[G(x) S(x)]

b. Three girls slept.
λPλQ[#(P ∩ Q) ≥ 3](G)(S)
x y z[G(x) G(y) G(z) S(x) S(y) S(z) ¬ x=y ¬ y=z ¬ x=z]

But others cannot be expressed in predicate logic, most notably sentences with NPs that have proportional quantifiers:

(26) Most girls slept.
λPλQ[#(P ∩ Q) > 1/2 #P](G)(S)

The problem with proportional quantifiers is that they have to refer to the cardinality of two *sets*. In our example, we need the number of girls, G. If we had that (say, the number of girls is n), then the sentence could be rendered just like “At least (n+1) girls slept”, and this we could express in first-order predicate logic. But first-order predicate logic does not allow us to use an operator like the cardinality operatior “#”: This is an operator that takes a *predicate* as an argument, and that’s not defined in first-order predicate logic. It can be defined in second-order predicate logic, of course, but second-order predicate logic does not have the nice logical feature of completeness.