Session Objectives

- Distinguish between predicate and propositional logic.
- Define wffs in predicate logic.
- Formalise English statements in predicate logic.
- Define prenex normal form and check if a statement is in this form.

Propositional Logic and Predicate Logic

- The propositional calculus treats propositions as 'atomic' (i.e., indivisible) and is unable to capture such arguments as:

  Mice like to eat cheese
  The moon is made of cheese
  Therefore mice like to eat the moon

- Predicate logic has greater expressive power than propositional logic. Instead of taking propositions as the basic building blocks, atomic formulae are built up from simpler constituents (called terms) and predicate symbols.

Predicates

- A predicate is a logical expression that expresses a relation.
- For example, the predicate `british(x)` where `x` is a free variable, represents the property of being British. Replacing `x` by a constant creates a proposition, which may be true or false. So `british(tonyBlair)` is true but `british(georgeBush)` is false.
- Predicate calculus is based on set theory in the sense that variables are constrained to belong to some domain (or set). For example, the domain for the predicate `british` is the set of human beings.
- A statement like `british(42)` is not just false but meaningless.

Applications of Predicate Logic

Formal Specification It is very important that specification of software systems and components is clear and precise so that every member of the development team understands exactly how subsystems are supposed to work. A formal specification uses set theory and predicate logic to define software.

Database Design A database contains objects which are defined by the values of certain fields. A field corresponds to a variable, and the relationships between fields correspond to predicates. A relational database defines relations between variables, and the process of normalisation uses predicate logic.

Logic Programming Given that all computations can be defined as logical operations (such as proof), it makes sense to use a programming language that is based on logical inference (i.e. the predicate calculus) to perform them. In practice, there are scaling issues (because many algorithms become rather complicated when expressed in predicate logic).

Syntax of Predicate Logic

Quantifier symbols:

Universal quantifier: $\forall x \phi(x)$ indicates that the formula is true for all values of the variable x. When we want to make the domain precise, we may write $(\forall x \in X)\phi(x)$, where X is the set that x is drawn from.

Existential quantifier: $\exists x \phi(x)$ indicates that the formula is true for some (i.e. at least one) value of the variable x. Again, we may use the notation $(\exists x \in X)\phi(x)$.

Syntax of Predicate Logic II

The other symbols are

logical connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$ and brackets.

constants which are terms with a fixed value.

variables which are terms that can take different values. We will give them names starting with an upper-case letter.

predicates name a relationship between terms.

functions name a mapping from terms to a single term.

Formula Definition

Terms

- Any variable or constant symbol is a term.
- If f is an m-ary function and t_1, t_2, ..., t_m are terms, then $f(t_1, t_2, \ldots, t_m)$ is a term.

Formulae

- $\perp$ is a formula.
- If r is an m-ary relation and t_1, t_2, ..., t_m are terms, then $r(t_1, t_2, \ldots, t_m)$ is a formula.
- If A and B are formulae and x is a variable, then the following are formulae:

$$(\neg A), (A \wedge B), (A \rightarrow B), (A \vee B), (\exists x)(A), (\forall x)(B).$$

$\perp$ and $r(t_1, t_2, \ldots, t_m)$ are called atomic formulae. They are equivalent to a propositional variable in propositional calculus.

Variable Binding

- In the formulae $\exists x p(x)$ and $\forall x q(x)$, the variable x is said to be bound and can no longer be instantiated or qualified again with $\exists$ or $\forall$.
- A variable which is not bound is said to be free.
- We cannot write $\exists Z(\forall Z p(x, Z))$ as the variable Z is already bound in the inner expression.
- In propositional logic, a formula is either true or false. However, the truth of the formula $x < 5$ depends on the value of x.
- This ambiguity is removed (or at any rate reduced) when the variables are bound. The statement $\exists x(x < 5)$ is true and the statement $\forall x(x < 5)$ is false (for the domain of the integers).

Formalising English

- This can only be an approximate process because of the very different natures of the two kinds of language. The range of expressiveness of English is much greater, but within its limits, a formal language is more precise.
- For instance, in English there are several different phrases that correspond to the quantifier $\forall$: 'for all', 'for each', 'for every', each of which suggests different nuances which are obliterated in the formal version.
- The first step is to identify occurrences of propositional connectives.
- Then identify the quantifiers and relations, and finish with functions and constants.
- As a general rule of thumb (but certainly not a rule that is always true), a universal quantifier is often followed by $\rightarrow$ and an existential quantifier by $\wedge$.

Examples

1. There is a bird that does not fly.

   We can rewrite this in a 'semi-formal' way as "There is a bird and that bird does not fly". This makes it clear that the main connective is 'and'. 'There is' is clearly translated to the $\exists$ quantifier. We need two one-place predicates `bird` and `flies`. The domain is the set of animals.

   $$\exists x(\text{bird}(x) \wedge \neg\, \text{flies}(x)$$

   Note that this formula follows the rule of thumb.

2. Everyone in the room spoke French or German.

   Here we can easily identify the universal quantifier ('every') and a disjunction ('or'). It is natural to introduce unary predicates room(x), french(x) and german(x). The domain is the set of people.

   $$(\forall x)(\text{room}(x) \rightarrow (\text{french}(x) \vee \text{german}(x))$$

Exercise

Every Conservative absentee was paired in the vote with a Labour MP.

Solution

Every Conservative absentee was paired in the vote with a Labour MP.

$$(\forall x)(\mathsf{con}(x) \rightarrow (\exists y)(\mathsf{lab}(y) \wedge \mathsf{pair}(x, y)))$$

Formalising Arithmetic for $\mathbb{N}$

We can define the base set with the constant term 0, the unary successor function S and the binary equality predicate equals. Examples of terms are

$$0, S(0), S(S(S(S(0))))$$

which represent the numbers 0, 1, and 4 respectively. The formula

$$\forall x \forall y [\text{equals}(S(x), S(y)) \rightarrow \text{equals}(x, y)]$$

represents the (true) statement that "if $S(x) = S(y)$ then $x = y$". We shall sometimes replace the equality predicate by the usual $=$ symbol.

Formal Definition of Addition

The binary addition function add can be defined by

$$\forall x(\text{equals}(\text{add}(x, 0), x))$$

$$(\forall x)(\forall y)(\text{equals}(\text{add}(x, S(y)), S(\text{add}(x, y))))$$

In more normal mathematical notation, this would be written as

$$\forall x(x + 0 = x)$$

$$\forall x \forall y((x + (y + 1))) = ((x + y) + 1))$$

In fact, from these two properties, it is possible to show all the usual properties of addition on natural numbers.

Exercise: Multiplication

Given the definition of addition above, write down in normal mathematical notation the following definition of the binary multiplication function mult:

$$\forall x(\text{equals}(\text{mult}(x, 0), 0))$$

$$(\forall x)(\forall y)(\text{equals}(\text{mult}(x, S(y)), \text{add}(\text{mult}(x, y), x)))$$

Formal Specification

- This idea of formalising systems using predicate logic will be taken much further in the final year module 'Formal System Development', where the Z language (essentially standard predicate logic with some useful extra predicates built in) will be used to define many different systems.
- One area where this has been used a lot in practice is for the definition of data structures (such as stacks, queues, etc.).
- Another formal language is the 'Vienna Development Method' (VDM). This is also used on practical specification applications.
- When I worked at Logica, I used Z to define a compiler (for most of the Pascal language omitting floating point arithmetic) for a client.

Solution

Given the definition of addition above, write down in normal mathematical notation the following definition of the binary multiplication function mult:

$$\forall x(\text{equals}(\text{mult}(x, 0), 0))$$

$$(\forall x)(\forall y)(\text{equals}(\text{mult}(x, S(y)), \text{add}(\text{mult}(x, y), x)))$$

$$(\forall x)x \times 0 = 0$$

$$(\forall x)(\forall y)x \times (y + 1) = (x \times y) + x$$

Session Objectives

- Distinguish between predicate and propositional logic.
- Define wffs in predicate logic.
- Formalise English statements in predicate logic.
- Define prenex normal form and check if a statement is in this form.