Predicate Calculus and Sets

Predicate Calculus

A **general purpose** language for describing objects, facts, and transformations for particular domains. Also called **First Order Logic**. It consists of

connective logic	{and, or, if, not, iff} inference, proof
object domains	{<unique atoms>}
quantification	{all.x, exists.x}
predicates	classes and properties
relations	True associations between objects
functions	indirect names, maps from one object to another

Predicate logic differs from propositional logic in two ways:

- Ground objects are more complex. Instead of being concepts which are either True of False, objects can consist of discrete elements from infinite sets. Functions are indirect names for particular objects. Relations are propositions about these complex objects.

- Quantification permits direct reference to sets of objects.

Compound Objects

propositions (Boolean variables)

The simplest objects. Has values in the carrier set {0,1}.

properties/domains (set variables)

Simple collections of objects.
Can have *finite* sets (computer science) or *infinite* sets (mathematics).

relations (pairs of objects)

Propositions between objects, relating one object to another.
Compound objects.

functions/procedures (structured pairs of objects)

Alternative names for objects ("2+3" is another name for 5).
Ways to move from one object to another.

Sets

Sets are unordered collections of unique objects.

`S = {x | <statement about x> }` *intensional*, set defined by a common property
`S = {a,b,c,...}` *extensional*, set defined by naming the members

characteristic function[x, S]:
A function which takes the value 1 exactly when `x` is in `S`.

membership: `x inS =def= x=s1 or x=s2 or x=s3 or ...`

empty set: `x not inS`

subset: `if x inS1, then x inS2`

union: `x inS1 or x inS2`

intersection: `x inS1 and x inS2`

difference: `x inS1 and not x inS2`

power set: `the set of all subsets of S`

recursive set membership:

```
x inS =def= not[x = empty-set] and
            x = first[S] or x in rest[S]
```

Set Axioms

Extent: `A = B iff x inA <--> x inB`

Specification: `Exists A. x inA iff x inB and P(x)`

Empty Set: The empty set is a member of all sets.

Specification means that a subset `A` of a set `A` can always be identified by specifying a property that uniquely identifies members of `A`.

Set Equivalence

Two finite sets are equal when they both contain exactly the same members.

```
S1 = S2 iff x inS1 -> x inS2 and x inS2 -> x inS1
```

Logic Plus Sets

Sets identify collections of objects.

Properties, or attributes, identify collections of objects.

The ***extrinsic description*** of a collection is an enumeration, or listing, of the objects in the set.

The ***intrinsic description*** of a collection is the name of the property which uniquely identifies the objects in a set.

```
S  =  {1,3,5,7,9}  =  OddDigit[x]
```

extrinsic *intrinsic*

Monadic Predicate Calculus

Adding properties to propositional logic makes *predicate calculus without relations*.

Monadic refers to operators with one argument only. I.e. Unary functions.

Quine (c. 1950) showed that, for finite sets,

monadic predicate calculus = propositional logic

One method to demonstrate this:

For each x in S, make (x in S) a proposition (i.e. True or False)

```
S  =  {a,b,c}  =  (a in S) & (b in S) & (c in S)
```

Quantification

Quantifiers introduce sets into logic, and serve to define the scope of variables in a logical expression.

Universal quantification: `All x. P(x)`

The statement `All x. P(x)` is True exactly when the predicate `P` (or the characteristic function for the set `P`) is True for all objects in the set `U` for which `x` is an arbitrary member.

For finite domains `U`, `All x. P(x) iff (x1 and x2 and...and xn)`

Existential quantification: `Exists x. P(x)`

The statement `Exists x. P(x)` is True exactly when the predicate `P` is True for at least one object in the set `U` for which `x` is an arbitrary member.

For finite domains `U`, `Exists x. P(x) iff (x1 or x2 or...or xn)`

Relationships between Quantifiers

All true = none false:

```
All x. P(x)  iff  (not (Exists x. (not P(x))))
```

All false = none true:

```
All x. (not P(x))  iff  (not (Exists x. P(x)))
```

Not all true = at least one false:

```
(not (All x. P(x)))  iff  Exists x. (not P(x))
```

Not all false = at least one true:

```
(not (All x. (not P(x))))  iff  Exists x. P(x)
```