Predicate Logic

(or First Order Logic (FOL))

Why isn't propositional logic sufficient?

Example: All birds fly. $\rightarrow$ p

What does $\neg p$ mean?

Propositional logic only tells us that its truth value is the opposite of p.

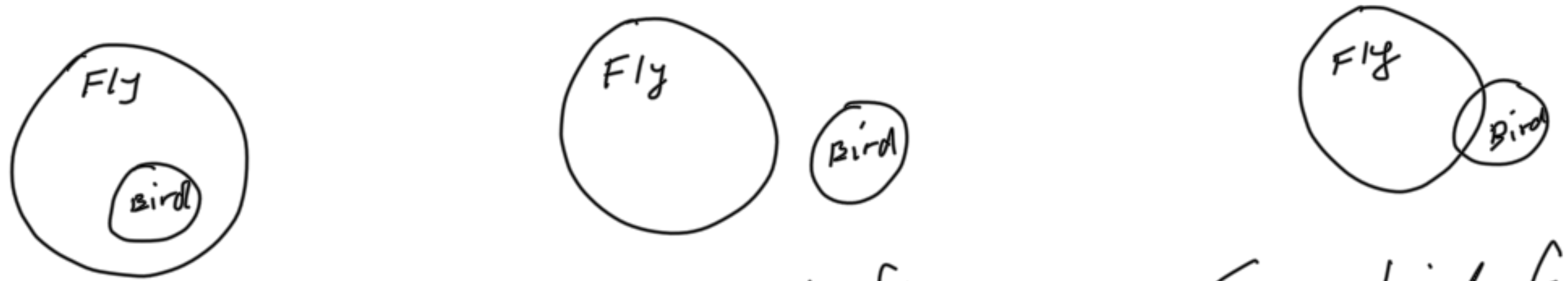
But what if we want to represent a more precise meaning of $\neg p$

i.e., Does it mean that No birds fly.

Does it mean that Some birds fly and some birds don't fly.

Since both of these sentences are propositions, and they don't mean the same thing, $\neg p$ can only represent one of them.

Let's use sets to describe these propositions.



All birds fly.

No birds fly.

Some birds fly and some birds don't fly.

To represent these different meanings we need to be able to talk about the members of these two sets.

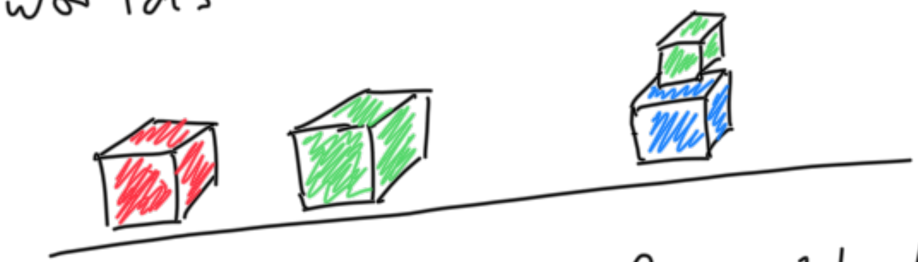
Predicate logic provides the representational tools to do this.

Informal Introduction to Predicate Logic

Like we had for propositional logic, we will have worlds that we want to describe.
But unlike the worlds for propositional logic, which were described by the truth or falsity of each proposition (truth valuations), here we want to talk about the objects in our worlds, properties of those objects, and relations among those objects.

Let's look at some worlds —

Blocks world



We have some blocks and a surface that they sit on

Blocks can be referred to by a name. (let's call our 4 blocks b1, b2, b3, b4)

Blocks can have a colour. (red, blue, green)

Blocks can be on top of other blocks. (b3 is on b4)

Note: "blocks" form a set of objects

We will use predicates to talk about sets, objects, relations

We will use constants to refer to objects.

Block(b1) Block(b2) Block(b3) Block(b4)

Red(b1) Green(b2) Green(b3) Blue(b4) On(b3, b4)

Onsurface(b1) Onsurface(b2) Onsurface(b4)

Arithmetic the natural numbers $\mathbb{N}$

$$1, 2, 3, 4, \ldots$$

So, first note that unlike the finite world in the Blocks World, we have a world with an infinite number of objects

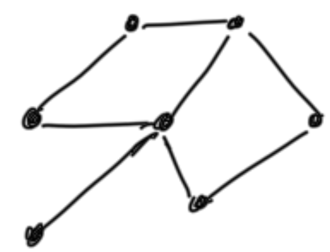
We can use the name of the number to refer to it.

We have relations such as less than.

Number(1) , Number(2), ...

Lessthan (1,2), Lessthan(2,3) , Lessthan(1,3) ...

Graphs



We have nodes, let's call them n1, n2, ... n7

Edges can be a relation — an edge connects two nodes.

Node(n1), Node(n2), ..., Node(n7)

Edge(n1, n2) Edge(n2, n3), Edge(n1, n4)

...

What do these worlds have in common?

We have sets of objects — called the <u>domain</u> or <u>universe</u>

constants - used to name the objects

predicates - used for properties (unary predicates) and relations (n-ary predicates $n>1$)

What we haven't seen are

functions (eg. succesor function is the arithmetic world — $+1$ or succ so it is another way to refer to objects (eg $\mathrm{succ}(1)$ is 2)

The other thing that may not be obvious is that the relations and objects in these worlds are more or less the same.

Blocks	Arithmetic	Graphs
b1, ..., b4 are blocks	1, 2, 3, ... are numbers	n1, ..., n7 are nodes
blocks can be on other blocks	numbers can be less than other numbers	nodes can be connected to other nodes

We will see shortly that when we interpret our formal language, these worlds — depending on what we want to represent — may be proper interpretations (these will be semantics)

Definitions

Universe (or domain) - a nonempty set

constant symbol - refers to an object in universe
(b1, b2, etc. ; 1, 2, 3, etc. ; n1, n2, etc.)

predicate (property or relation) - represents a property that an object or collection of objects may have
(Blue(b4), On(b3, b4), Lessthan(1, 2), Edge(n1, n2))

function - another way to refer to an object
(succ(3) refers to 4)

Whether we use a predicate or a function to refer to a property or a relation is arbitrary.

It's use is not:

succ(3) refers to 4

Successor(3,4) is a relation between 3 and 4 variable – used to refer to a member of a set without naming it

quantifier – used to refer to objects in the universe

universal quantifier – refers to all objects in the universe

existential quantifier – refers to some objects in the universe

Example: We have seen Block(b1) which says "b1 is a Block"

We can then also have Block(x) which says

"some object is a Block" (without naming the object)

We can then use the quantifiers to talk about collections of objects without naming the objects in these collections

Examples

(we will introduce the formal representation later)

- for every object in our universe that can take the place of the variable x $\qquad S(x)$

  which can be read as "every object in our universe is an S"

- for some object(s) in our universe that can take the place of the variable x $\qquad S(x)$

  which can be read as "some object(s) is (are) an S"

Notation

constants - lower case letters and numbers

a, b, c, b1, b2, n1, n2, 1, 2, 3

predicates - begins with upper case letter

Block, Red, LessThan, Edge, P, Q

variables - lower case letters near the end of alphabet

w, x, y, z

functions - lower case letters near middle of alphabet

f, g or words succ

Notation (continued)

connectives – $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$

(these have the same meaning as in propositional logic)

quantifiers – $\forall$ (universal quantifier)

$\exists$ (existential quantifier)

punctuation – ( ) ,

Beginning of Syntax

In the informal introduction we were introduced to some syntax - let's formalize this

Let's start with

formula : predicate$(\text{term}_1, \text{term}_2, \dots, \text{term}_n)$

if the predicate is unary it requires 1 term
if the predicate is binary it requires 2 terms
if the predicate is n-ary it requires n terms

predicate : P, Q, S, Block, ...

term : constant : a, b, c, b1, n3

variable : x, y, ...

function : $f(\underbrace{\qquad\qquad}_{\text{of the correct arity}})$

$\underbrace{\text{succc}}_{\text{unary function}}(3)$

So, with the current definition of "formula" we can write

$P(a)$ "P" is a predicate "a" is a constant

$P(x)$ " "x" is a variable

$Q(a, x)$ Q is a binary predicate

$Q(f(a), g(b))$ "

f is a function a, b are constants
g is a function

$Q(f(x), g(y))$

Let's add some more syntactic possibilities for "formula"

formula : quantifier variable formula

quantifier : $\forall$

$\exists$

So, now we can write

$\forall x \; P(x)$

$\exists x \; P(x)$

$\exists x \; Q(a, x)$

$\exists x \; Q(f(x), b)$

$\forall y \, \exists x \; Q(x, y)$

Let's continue to add more syntactic possibilities for "formula"

formula : formula connective formula

$\neg$ formula

(formula)

connective : $\wedge$

$\vee$

$\rightarrow$

$\leftrightarrow$

So this let's us write formulas like

$$\forall x\ P(x) \wedge Q(x)$$

$$\exists x\ P(x) \wedge Q(b, f(x))$$

$$\exists x\ P(x) \wedge \exists x(b, f(x))$$

These three formulas point to a problem that needs to be solved: the <u>scope</u> of a quantifier

The scope of a quantifier depends on the smallest formula to the right of the quantifier

[so the definition of "formula" is very important]

$$\overbrace{\forall x\, \underbrace{\underbrace{P(x)}_{\text{formula}}}_{\text{scope of } \forall x}}^{\text{formula}} \qquad \overbrace{\overbrace{\forall x\, \underbrace{\underbrace{P(x)}_{\text{formula}}}_{\text{scope of } \forall x}}^{\text{formula}} \land \overbrace{Q(f(x), b)}^{\text{formula}}}^{\text{formula}}$$

If you want $\forall x$ to scope all the way to the end of the complete formula we need to create the correct formula

Remember the punctuation ( ) around a formula makes a formula

$$\overbrace{\forall x \underbrace{\underbrace{\left(\underbrace{\underbrace{P(x)}_{\text{formula}} \land \underbrace{Q(f(x), b)}_{\text{formula}}}_{\text{formula}}\right)}_{\text{formula}}}_{\text{scope of } \forall x}}^{\text{formula}}$$

One last item:

If a variable is in the scope of multiple quantifiers for that variable the variable is quantified by the most "inside" quantifier

quantified by this quantifier

quantified by this quantifier

$$\forall x \underbrace{\left(P(x) \land \exists x \underbrace{Q(f(x), b)}_{\text{scope of } \exists x} \right)}_{\text{scope of } \forall x}$$

Free and bound variables

A variable is bound by a quantifier with the same variable if the variable is in the scope of that quantifier.

A variable is free otherwise.

FOL allows free and bound variables in formulas.

Here, we will only consider formulas with bound variables only.

These formulas are called <u>sentences</u>.

Now, we will look at the semantics (meaning) of sentences.