# 4. Classical First-Order Predicate Logic

# Why Predicate logic?

It is a powerful extension of propositional logic. It is the most important logic of all.

Propositional logic is quite nice, but not very expressive.

Statements like

- the list is ordered
- every worker has a boss
- there is someone worse off than you

need something more than propositional logic to express.

Propositional logic cannot express arguments like this one of De Morgan:

- A horse is an animal.
- Therefore, the head of a horse is the head of an animal.

# Predicate logic in a nutshell

Predicate logic is concerned with describing *relationships between objects*, and the ways in which different relationships are logically connected. So, atomic formulas become *structured*.

- Syntactically, there are *6 new features*:
  1. *Predicates* (that take *arguments*): $\texttt{sister}(\texttt{Bob}, \texttt{Mary}), \ldots$.
  2. *Constants*: `Bob`, `Room_308`, ....
  3. *Variables*: $x, y, z, \ldots$.
  4. *Quantifiers*: $\forall$ (*for all*); $\exists$ (*there exists*)
  5. *Functions*: `father_of(_)`, `sum_of(_,_)`, $+, -, \times, \ldots$
  6. *Equality*: $=$

- Semantically, the notion of *situation* in predicate logic is more complex than in propositional logic. We have to give meaning to constants, functions, predicates and variables.

# 4.1 Syntax — The formal language

First, we need to fix a precise definition of the formal language, that is the *syntax* of predicate logic.

# Splitting the atoms - new atomic formulas

In propositional logic, we regard phrases like *Arron is a student* and *Arron and Russell are friends* as atomic, without an internal structure.
Now we look inside!

We regard "being a student" as a *property* that Arron (and other objects) may, or may not, have; "being friends" as a *relation* that Arron and Russell (and other two objects) may, or may not, have.

So we introduce:

- *Predicate symbols*, to describe properties of and relations between objects.
  - `student`. It takes 1 argument – it is *unary*, or its 'arity' is 1.
  - `friends`. It takes 2 arguments – it is *binary,* or its 'arity' is 2.
- *Constants,* to name objects
  - `Arron`, `Russell`, . . .,

Then `student(Arron)` and `friends(Arron,Russell)` are examples of new *atomic formulas*.

# Quantifiers

You may think that writing `friends`(`Arron`, `Russell`) is not much more exciting than what you did in propositional logic, writing `Aron and Rusell are friends`.

But what about the phrase *Arron is a friend of everyone*?

Predicate logic has a machinery to vary the arguments to `friends`.

This allows us to express characteristics about the relationship 'friends'.

The machinery is called *quantifiers.*
(The word was introduced by De Morgan.)

# What are quantifiers?

A quantifier specifies a quantity (of objects that have some property).

- *All* students work hard.
- *Some* students are asleep.
- *Most* lecturers are crazy.
- *Eight out of ten* cats prefer it.
- *No one* is worse off than me.
- *At least six* students are awake.
- *There are infinitely many* prime numbers.
- *There are more* PCs *than* there are Macs.

# Quantifiers in predicate logic

There are just two:

- $\forall$ (or `(A)`): 'for all'
- $\exists$ (or `(E)`): 'there exists' (or 'some')

Some other quantifiers can be expressed with these. (They can also express each other.)

But quantifiers like *infinitely many* and *more than* cannot be expressed in first-order logic in general. (They can in, e.g., second-order logic. And even first-order logic can sometimes express them in special cases.)

How do they work?

> We've seen expressions like `Russell`, `Arron`, etc. These are *constants*, like $\pi$, or $e$. So, to express 'Arron is a friend of everyone' we need *variables* that range over all people, not just Russell, etc.

# Variables

We use *variables* to do quantification. We fix an infinite collection (or 'set') $V$ of variables: e.g., $x, y, z, u, v, w, x_0, x_1, x_2, \ldots$
(Sometimes I write $x$ or $y$ to mean 'any variable'.)

So can write formulas like $\texttt{student}(x)$.

- Now, to say 'Everyone is a student', we'll write $\forall x\, \texttt{student}(x)$.
  This is (literally) read as: 'For all $x$, $x$ is a student'.
- 'Someone is a student', can be written as $\exists x\, \texttt{student}(x)$.
  'There exists $x$ such that $x$ is a student.'
- 'Frank has a student friend', can be written

$$\exists x(\texttt{student}(x) \wedge \texttt{friend}(\texttt{frank}, x)).$$

  'There is an $x$ such that $x$ is a student and $x$ is a friend of Frank.'
  Or: 'For some $x$, $x$ is a student and $x$ is a friend of Frank.'

# Function symbols

In arithmetic (and Haskell) we are used to *functions,* such as $+, -, \times, \sqrt{x}, ++$, etc.

Predicate logic can do this too.

- *A function symbol refers to an object in terms of another object or objects.*

A function symbol comes with a fixed arity (number of arguments). Examples of functions: `father_of`(_), `mother_of`(_), `sum_of`(_, _).

# Equality

$=$ is a particularly important predicate symbol, also used widely outside of mathematics:

- $\texttt{t}_1 = \texttt{t}_2$ means "$t_1$ and $t_2$ refer to the same object".

We now make all of this precise.

# Signatures

## Definition 4.1 (signature)

The *signature* of a predicate logic is a triple $\langle \mathcal{K}, \mathcal{F}, \mathcal{P} \rangle$, where $\mathcal{K}$ is a (possibly empty) set of constants, $\mathcal{F}$ is a (possibly empty) set of function symbols, and $\mathcal{P}$ is a set of predicate symbols. Function and predicate symbols have specific arities.

Some call it a *vocabulary,* or (loosely) *language.*

It replaces the collection of propositional atoms you have seen in propositional logic.

We usually write $L$ to denote a signature. We often write $c, d, \ldots$ for constants, $f$, $g$ for function symbols and $P, Q, R, S, \ldots$ for predicate symbols.

# Example of a simple signature

Which symbols we put in $L$ depends on what we want to say.

For illustration, we'll use a handy signature $L$ consisting of:

- constants `frank`, `susan`, *tony*, `heron`, `clyde`, and `c`
- unary function symbol `father_of` (arity 1)
  (also `father_of`/1)
- unary predicate symbols `PC`, `Human`, `Lecturer` (arity 1)
  (also `PC`/1, `Human`/1, `Lecturer`/1)
- a binary predicate symbol `Bought`(arity 2).
  (also `Bought`/2)

**Warning:** things in $L$ are just symbols — syntax. They don't come with any meaning. To give them meaning, we'll need to work out (later) what a *situation* in predicate logic should be.

# Terms

To write formulas, we need *terms* to name objects. Terms are not formulas. They will not be true or false.

## Definition 4.2 (term)

Fix a signature $L$.

1. Any constant in $L$ is an $L$-term.
2. Any variable is an $L$-term.
3. If $f$ is an $n$-ary function symbol in $L$, and $t_1, \ldots, t_n$ are $L$-terms, then $f(t_1, \ldots, t_n)$ is an $L$-term.
4. Nothing else is an $L$-term.

A *closed term* or (as computing people say) *ground term* is one that doesn't involve a variable.

Examples

**frank**, **father_of**(**susan**) (ground terms).

$x$, $\mathtt{g}(x)$, $\mathtt{father\_of}(\mathtt{g}(y))$ (not ground terms).

# Formulas of first-order logic

## Definition 4.3 (formula)

Fix a signature $L$.

1. If $R$ is an $n$-ary predicate symbol in $L$, and $t_1, \ldots, t_n$ are $L$-terms, then $R(t_1, \ldots, t_n)$ is an atomic $L$-formula.
2. If $t, t'$ are $L$-terms then $t = t'$ is an atomic $L$-formula. (Equality — very useful!)
3. $\top, \bot$ are atomic $L$-formulas.
4. If $\phi, \psi$ are $L$-formulas then so are $(\neg\phi)$, $(\phi \wedge \psi)$ $(\phi \vee \psi)$, $(\phi \to \psi)$, and $(\phi \leftrightarrow \psi)$.
5. If $\phi$ is an $L$-formula and $x$ a variable, then $(\forall x\, \phi)$ and $(\exists x\, \phi)$ are $L$-formulas.
6. Nothing else is an $L$-formula.

# Atomic formulas, Literals, and Sentences

- An *atomic formula* is a predicate symbol with arguments filled in with terms.
  - $\texttt{Lecturer}(\texttt{susan})$
  - $\texttt{PC}(x)$
- A *literal* is an atomic formula or its negation:
  - $\texttt{Sum\_of}(x, \texttt{4}) = \texttt{10}$
  - $\neg\texttt{Lecturer}(\texttt{mother\_of}(x))$
- A *sentence* is a formula with all variables in the *scope* of a quantifier.
  - $\forall x \forall y \texttt{Bought}(x, y)$

**Binding conventions:** as for propositional logic, plus: $\forall x, \exists x$ have same strength as $\neg$.

Formation trees, sub-formulas and clauses can be done much as before.

# Examples of formulas

1. $\texttt{Bought}(\texttt{frank}, x)$

2. $\neg\texttt{Bought}(\texttt{frank}, x)$

3. $\exists x\, \texttt{Bought}(\texttt{frank}, x)$

4. $\forall x(\texttt{Lecturer}(x) \rightarrow \texttt{Human}(x))$

5. $\forall x(\texttt{Bought}(\texttt{tony}, x) \rightarrow \texttt{PC}(x))$

# Examples of formulas cont.

6. $\forall x(\texttt{Bought}(\texttt{father\_of}(\texttt{tony}), x) \rightarrow \texttt{Bought}(\texttt{susan}, x))$

# Examples of formulas cont.

6. $\forall x(\texttt{Bought}(\texttt{father\_of}(\texttt{tony}), x) \to \texttt{Bought}(\texttt{susan}, x))$
   'Susan bought everything that Tony's father bought.'

7. $\forall x\, \texttt{Bought}(\texttt{father\_of}(\texttt{tony}), x) \to \forall x\, \texttt{Bought}(\texttt{susan}, x)$
   'If Tony's father bought everything, so did Susan.'
   *Note the difference!*

8. $\forall x \exists y\, \texttt{Bought}(x, y)$
   'Everything bought something.'

9. $\exists y \forall x\, \texttt{Bought}(x, y)$
   'There is something that everything bought.'
   *Note the difference!*

10. $\exists x \forall y\, \texttt{Bought}(x, y)$
    'Something bought everything.'

You can see that predicate logic is rather powerful — and terse.