10: LOGIC PROGRAMMING

COMP702: CLASSICAL ARTIFICIAL INTELLIGENCE

LOGIC

PREDICATES

- A **predicate** is a **function** which returns a **Boolean**
- In logic, a predicate is a **parameterised statement** which is **true or false**
- E.g. `LivesIn(Bob, Falmouth)` could be a predicate representing the statement "Bob lives in Falmouth"
- Two parameters: *Bob* and *Falmouth*

QUANTIFIERS

- Let $P(x)$ be a predicate
- $\forall x : P(x)$ means that $P(x)$ is true for all values of x
- $\exists x : P(x)$ means that there exists at least one value of x such that $P(x)$ is true

IMPLICATION

- "A implies B" means "if A is true then B is true"
- Written as $A \Rightarrow B$
- E.g. if someone lives in Falmouth, then they live in Cornwall
- $\forall x : (\texttt{LivesIn}(x, \texttt{Falmouth}) \Rightarrow \texttt{LivesIn}(x, \texttt{Cornwall}))$

EQUIVALENCE

- If $A \Rightarrow B$ and $B \Rightarrow A$, then A is true **if and only if** B is true
- This means A and B are **logically equivalent**
- Written as $A \Leftrightarrow B$

PROLOG

PROLOG

- A **declarative language** for **logic programming**
- First developed in early 1970s
- Several implementations available today – I'm using SWI-Prolog (which is free open source)

USES OF PROLOG

- Well suited for declarative propositional logic
- Can do constraint programming
- Can do STRIPS-like planning
- Backtracking search comes built in

ANATOMY OF A PROLOG PROGRAM

- A standard Prolog program specifies:
  - **Facts** – these are predicates which are true
  - **Rules** – implications of the form `Head :- Body`, which states that `Head` is true if `Body` is true
- A program by itself doesn't do anything – we run **queries** against it
  - Is a given predicate true?
  - For what parameter values is a given predicate true?

A FACT

`lives_in` is a predicate. Predicates are introduced when first used.

```
lives_in(bob, falmouth).
```

bob and `falmouth` are atoms – named variable values. Atoms begin with lowercase letters.

QUERIES

- We can query predicates for truth or falsehood
  - `?- lives_in(bob, falmouth).`
    - `true.`
  - `?- lives_in(bob, penryn).`
    - `false.`

QUERIES

- We can also introduce variables – Prolog gives us all possible values of the variable for which the predicate is true
  - `?- lives_in(bob, X).`
    - `X = falmouth.`
  - `?- lives_in(Y, falmouth).`
    - `Y = bob.`
  - `?- lives_in(Z, penzance).`
    - `false.`

Variables begin with uppercase letters (to distinguish them from atoms)

There are no variable values that make the predicate true, so Prolog returns false

RULES

:- can be read as ⇐ (right hand side implies left hand side)
If RHS is true, then LHS is true

```
lives_in(X, cornwall) :- lives_in(X, falmouth).

drinks_cider(X) :- lives_in(X, cornwall).
```

When variables appear in rules, there is an implicit ∀ quantifier – i.e. this implication holds true for all X

QUERYING RULES

- We can run queries based on the chains of implications in our rules
- `?- lives_in(bob, cornwall).`
  - `true.`
- `?- drinks_cider(X).`
  - `X = bob.`

PATTERN MATCHING

- We can define multiple rules with the same predicate, or even the same LHS
- When searching in response to a query, Prolog applies whichever one matches

CONJUNCTION

- The RHS of a rule can contain a conjunction (an AND)
- Denoted by predicates separated by commas

```
is_local_student(X) :- lives_in(X, cornwall),
studies_at(X, falmouth).
```

HOW DOES PROLOG EXECUTE QUERIES?

- Uses **Selective Linear Definite (SLD) resolution**
- Multiple rules for a predicate imply a **choice point**, which leads to a **backtracking search**
- Care is needed to avoid **infinite loops** etc
- Can be thought of as a special case of a **Boolean satisfiability problem (SAT problem)**
  - In the general case, SAT is NP-complete

CONSTRAINT PROGRAMMING IN PROLOG

CONSTRAINT PROGRAMMING

- Prolog can do constraint programming with the aid of libraries
- Several available; e.g. `clpfd` allows for constraint programming with integer maths
- `:- use_module(library(clpfd)).`
- Introduces new relational operators #=, #<, #<= etc, which can be used for constraint solving

SOLVING EQUATIONS

- `X #= 1 + 2.`
  - `X = 3.`
- `15 #= 3 * Y.`
  - `Y = 5.`
- `21 #= X*Y, X #> 1, Y #= 1, X #< Y.`
  - `X = 3,`
  - `Y = 7.`

MORE EXAMPLES

- https://www.swi-prolog.org/man/clpfd.html