# Artificial Intelligence with Logic Programming - Notes

Dom Hutchinson

January 28, 2020

## Contents

# 1 Introduction

**Definition 0.1 -** *Types of AI*

1. *Weak AI* - Can solve a specific task.
2. *Strong AI* - Can solve general problems.
3. *Ultra Strong AI* - Can solve general problems & explain why/what it is doing.

# 2 Logic Programming

**Definition 0.1 -** *Logic Programming*
*Logic Programming* is a *declarative paradigm* where programs are concieved as a logical theory, rather than a set-by-step description of an algorithm. A Procedure call is viewed as a theorem which the truth needs to be established about. (*i.e.* executing a programming is analogous to searching for truth in a system).

**Remark 0.1 -** *Variables*
In *Logic Programming* a *Variable* is a variable in the mathematical sense, that is they are are placeholders that can take on any value.

**Remark 0.2 -** *Machine Model*
A *Machine Model* is an abstraction of the computer on which programs are executed. In *Imperative Programming* we assume a dynamic, state-based machine model where the state of the computer is given by the contents of its memory & a program statement is a transition from one statement to another. In *Logic Programming* we do not assume such a dynamic model.

## 2.1 Clausal Logic

**Notation 1.1 -** *Variables & Values*
*Variables* are denoted by having a capitalised first letter, whereas *values* are completly lowercase.

**Definition 1.1 -** *Clausal Logic*
*Clausal Logic* is a formalism for representing & reasoning with knowledge.

| **Keyword** | **Description** |
|---|---|
| `S:-C` | **If** condition `C` holds **then** statement `S` is true. |
| `S:-C1,C2` | **If** conditions `C1` **and** `C2` both hold **then** statement `S` is true. |
| `connected(X,Y,...)` | Objects are connected to each other. |
| `nearby(X,Y,...)` | Objects are near to each other. |

*N.B.* We define *connected*, *close*, etc. depending upon the problem scenario.

**Definition 1.2 -** *Facts & Rules*
*Facts* are logical formulae which are defined for explicit values **only**. *Facts* denoted unconiditional truth.

```
nearby(bond_street,oxford_circus).
```

*Rules* are logical formula which are defined in termms of variables (and explicit values). *Rules* denote conditional truth.

```
nearby(X,Y):-connected(X,Z,L),connected(Z,Y,L)
```

**Definition 1.3 -** *Query,* `?-`
A *Query* asks a question about the knowledgebase we have defined. If we just pass *values* to a *Query* then it shall simply return whether the statement is true or not. If we pass *unbound variables* as well then it shall return values for the variable which make the statement true, if any exist.

**Example 1.1 -** *Query*

```
1 ?-nearby(bond_street,oxford_circus)
2 ?-nearby(bond_street,X)
```

(1) will return *true* if we have defined `bond_street` to be near to `oxford_circus`.
(2) will return all the values of `X` (*i.e.* stations) which are near to `bond_street`.

**Definition 1.4 -** *Resolution*
In order to answer a query `?-Q1,Q2,...` find a rule `A:-B1,...,Bn` such that `A` matches with `Q1` then proceed to answer `?-B1,...,Bn,Q2,...`.
This is a *procedural interpretation* of logical formulae & is what allows *Logic* to be a programming language.

**Definition 1.5 -** *Functor*
*Functor*s provide a way to name a complex object composed of simpler objects & are never evaluated to determine a value.

```
reachable(X,Y,noroute):-connected(X,Y,L)
reachable(X,Y,route(Z,R)):-connected(X,Z,L),connected(Z,Y,R)
```

Querying `?-reachable(oxford_circus,tottenham_court_road,R)` will return a route `R` which connects the two stations, on a single line.
The above definition can be read as `X` is reachable from `Y` if they are connected **or** if there exists a station `Z` which is connected.

**Definition 1.6 -** *List Functor,* `.`
The *List Functor* takes two arguments, one on each side, and has terminator `[]`.

[a,b,c]≡`.(a,.(b,.(c,[])))`

Alternatively we can use a pipe to distinguish between a value and the rest of the list

`[X,Y|R]`