

Rules and Conjunctions

A man is happy if he is rich and famous
might translate to:

`happy(Person) :-`

```
    man(Person) ,  
    rich(Person) ,  
    famous(Person) .
```

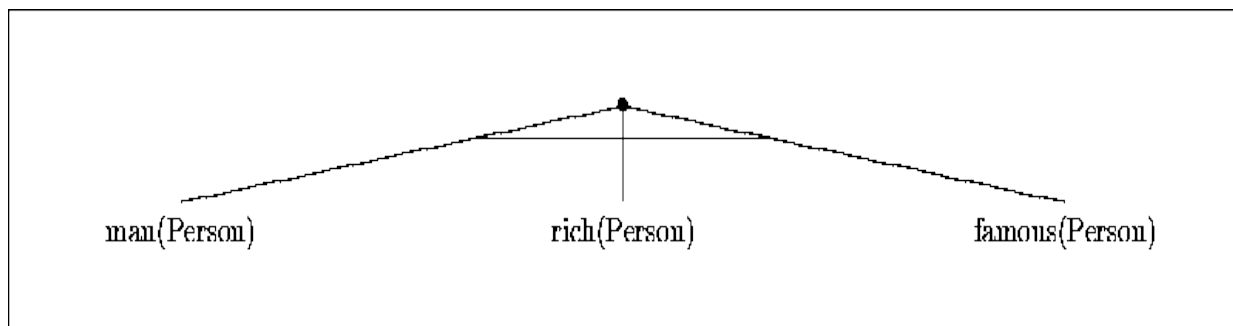
The `,` indicates the conjunction and is roughly equivalent to the \wedge of predicate calculus. Therefore, read `,` as 'and' \diamond . The whole of the above is one (non-unit) single clause.

It has three subgoals in its body ---these subgoals are 'conjoined'.

In this single clause, the logical variable **Person** refers to the same object throughout.

By the way, we might have chosen any name for the logical variable other than **Person**. It is common practice to name a logical variable in some way that reminds you of what kind of entity is being handled.

We now describe this clause graphically. In this case, we are going to represent conjunctions using an AND tree. Here is an AND tree that represents the above.



The way in which we discriminate between an OR tree and an AND tree is the use of a horizontal bar to link the subgoals. We need this distinction because we are going to represent the structure of a program using a combined AND/OR tree.

Exercise 2.4 *A few more exercises. Each of these statements should be turned into a rule (non-unit clause) with at least two subgoals —even though some statements are not immediately recognisable as such:*

1. *you are liable to be fined if your car is untaxed*
2. *two people live in the same house if they have the same address*
3. *two people are siblings if they have the same parents*

1. **liable_for_fine(X):- owns_car(X,Y), untaxed(Y).**

We assume that **liable_for_fine/1** holds when its argument is (a person) liable for a fine, that **owns_car/2** holds when the first argument possesses the object named in the second argument (and this object is a car), and that **untaxed/1** holds for all those objects that are required by law to be taxed and are not!

2. **same_house(X,Y):- address(X,Z), address(Y,Z).**

The **same_house/2** relation holds between two arguments (people) if the **address/2** relation holds between one of these arguments and a third object and between the other and the same third object.

Note that this makes **same_house(fred,fred)** true.

3. **siblings(X,Y):- mother(X,M), mother(Y,M), father(X,P), father(Y,P).**

The **siblings/2** relation holds between the two arguments when each is related via the **mother/2** relation to a common object and via the **father/2** relation to a (different) common object.

This is not correct if the intended meaning is to prevent one person being their own sibling. We would revise this by adding a subgoal such as **not_same(X,Y)**.

Note that we could have designed a **parents/3** predicate (relation) such that, for example, the second argument is the mother and the third is the father of the first argument. This would result in **siblings(X,Y):- parents(X,M,P), parents(Y,M,P)**.

Rules and Disjunctions

Someone is happy if they are healthy, wealthy or wise.

translates to:

`happy(Person) :-`

healthy(Person) .

happy(Person) :-

 wealthy(Person) .

happy(Person) :-

 wise(Person) .

Note how we have had to rewrite the original informal statement into something like:

Someone is happy if they are healthy OR
Someone is happy if they are wealthy OR
Someone is happy if they are wise

We have also assumed that each clause is (implicitly) universally quantified. *i.e.* the first one above represents $\forall X.(\text{healthy}(X) \Rightarrow \text{happy}(X))$.

The predicate name 'happy' is known as a *functor*. (Any predicate name is a functor).

The functor **happy** has one *argument*.

We describe a predicate with name 'predname' with arity 'n' as **predname/n**. It has one argument ---we say its *arity* is 1.

The predicate **happy/1** is defined by three clauses.

Exercise 2.5 *Each of these statements should be turned into several rules:*

1. *you are british if you are welsh, english, scottish or northern irish*
 2. *you are eligible for social security payments if you earn less than £ 28 per week or you are an old age pensioner*
 3. *those who play football, rugger or hockey are sportspeople*
1. **british(X):- welsh(X).**
british(X):- english(X).
british(X):- scottish(X).
british(X):- northern_irish(X).

Note that we have preserved the order of nationalities as described in the statement. This has no logical significance.

2. **eligible_social_security(X):- earnings(X,Y), less_than(Y,28).**
eligible_social_security(X):- oap(X).

In the first part of the disjunction, we have introduced an additional predicate **less_than/2** which has the reading that the relation holds when the first argument is less than the second.

Also, note that the original statement does not make it clear whether or not someone could qualify both as an old age pensioner (oap) and as someone earning very little. This could become an important issue.

3. **sportsperson(X):- plays(X,football).**
sportsperson(X):- plays(X,rugger).
sportsperson(X):- plays(X,hockey).

Conjunctions and Disjunctions

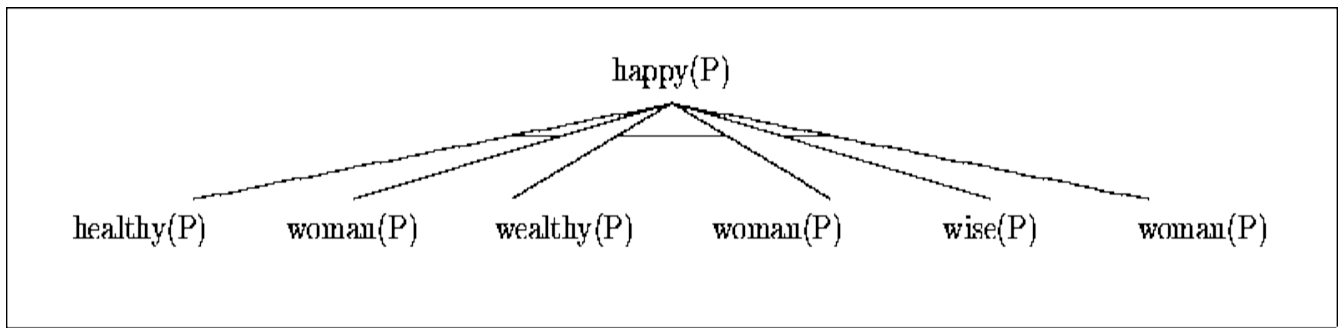
We are now ready for the whole thing: let us go back to the set of rules as found in section [2.12](#) and some basic facts.

Program Database	
woman(jean).	
woman(jane).	
woman(joan).	
woman(pat).	
wise(jean).	
wealthy(jane).	
wealthy(jim).	
healthy(jim).	
healthy(jane).	
healthy(jean).	
happy(P):-	healthy(P), woman(P).
happy(P):-	wealthy(P), woman(P).
happy(P):-	wise(P), woman(P).

and consider the solution of the goal

happy(jean)

Here is the standard AND/OR tree representation of the search space again:



and the goal succeeds.

Note that

1. Both the subgoal **healthy(jean)** and **woman(jean)** have to succeed for the whole goal to succeed.
2. We then return to the top level.

Now consider the top level goal of

happy (joan)

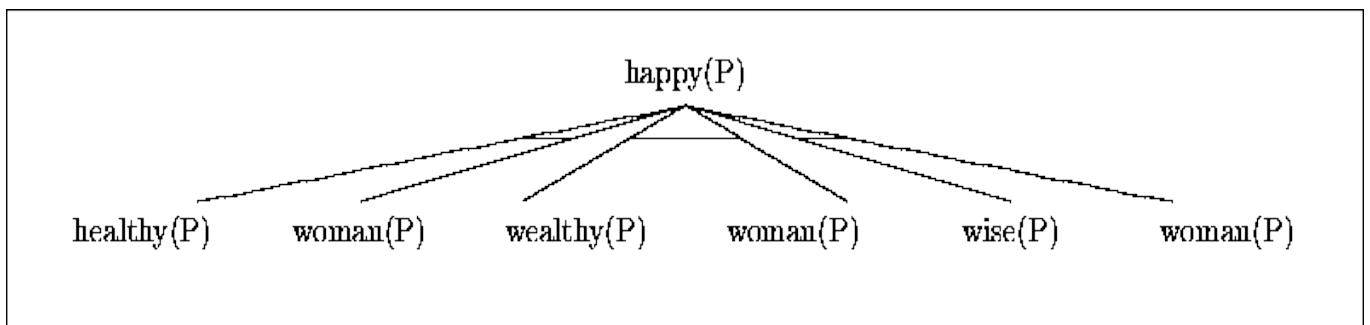
The resolution process generates the subgoals **healthy(joan)** and **woman(joan)** from the first clause for **happy/1**. In all, **Prolog** tries three times to match **healthy(joan)** as there are three clauses for **healthy/1**. After failing **healthy(joan)**, however, **Prolog** does not try to solve **woman(joan)** ---there is no point in doing so.


There is another way of trying to prove **happy(joan)** using the second clause of **happy/1**. The resolution process again generates subgoals --- **wealthy(joan)** and **woman(joan)**--- and **wealthy(joan)** fails. A third attempt is made but this founders as **wise(joan)** fails. Now back to top level to report the complete failure to satisfy the goal.

Now consider

happy (P)

as the top level goal.



Much more complicated. First, **healthy(P)** succeeds *binding* **P** to **jim** (*P/jim*) but when the conjunctive goal **woman(jim)** is attempted it fails. **Prolog** now *backtracks* . It reverses along the path through the tree until it can find a place where there was an alternative solution.

Of course, **Prolog** remembers to *unbind* any variables exactly at the places in the tree where they were bound.

In the example we are using we again try to resolve the goal **healthy(P)** ---succeeding with **P** bound to **jane**. Now the conjunction can be satisfied as we have **woman(jane)**. Return to top level with **P** bound to **jane** to report success. What follows is what appears on the screen:

```
?- happy (P) .
```

```
P=jane
```

```
yes
```

Prolog offers the facility to *redo* a goal ---whenever the top level goal has succeeded and there is a variable binding. Just type ``;" followed by RETURN ---``;" can be read as *or*. If possible, **Prolog** finds another solution. If this is repeated until there are no more solutions then we get the sequence of solutions:

```
jane  
jean  
jane  
jean
```

It is worth trying to verify this.

Basically, trying to follow the behaviour of **Prolog** around the text of the program can be very messy. Seeing how **Prolog** might execute the search based on moving around the AND/OR tree is much more coherent *but* it requires some effort before getting the benefit.

What You Should Be Able To Do

After finishing the exercises at the end of the chapter:

You should be able to represent any simple fact in legal **Prolog**.
You should be able to split up a disjunctive expression into a set of **Prolog** clauses.
You should be able to express a simple conjunctive expression as a single clause.
You should be able to represent most rules in legal **Prolog**.

There is no perfect solution to the problem of representing knowledge. You may generate representations that differ wildly from someone else's answers. To find out which answer is best and in what context will require some deeper thought.

Exercise 2.6 *Here is a small set of problems that require you to convert propositions into Prolog clauses. Make sure you explain the meaning of your representation:*

1. $a \Rightarrow b$
2. $a \vee b \Rightarrow c$
3. $a \wedge b \Rightarrow c$
4. $a \wedge (b \vee c) \Rightarrow d$
5. $\neg a \vee b$

Exercise 2.7 *A simple collection of problems. Represent each statement as a single Prolog clause:*

1. *Billy studies AI2*
2. *The population of France is 50 million*
3. *Italy is a rich country*
4. *Jane is tall*
5. *2 is a prime number*
6. *The Welsh people are British*
7. *Someone wrote Hamlet*
8. *All humans are mortal*
9. *All rich people pay taxes*
10. *Bill takes his umbrella if it rains*
11. *If you are naughty then you will not have any supper*
12. *Firebrigade employees are men over six feet tall*

1. `studies(bill,ai2).`

We have revised 'AI2' to 'ai2'. We could have simply put quotes around as in `studies(bill, 'AI2 ')`.

2. **population(france,50).**

where the reading is that the population of the first object in the relation **population/2** is the second object expressed in millions of people.

Note we have changed 'France' to 'france'.

3. **rich_country(italy).**

Here, the statement has been expressed as a unary 'relation' of something being a rich country.

4. **height(jane,tall).**

We have covered a similar example previously.

5. **prime(2).**

We have asserted that the attribute of primeness belongs to the number 2.

6. **british(X):- welsh(X).**

The statement has been turned into the equivalent 'everybody who is welsh is british'. This is an alternative to the statement **subset(welsh,british)**. We read this as meaning that the **subset/2** relation holds between the set of welsh people and the set of british people.

As usual, we have lower-cased the words 'Welsh' and 'British'.

7. **author(hamlet,someone).**

This is a trick question. You cannot answer this one from the notes. Why not? Well, let me give the meaning of the above: the **author/2** relation holds between 'hamlet' (which stands for the famous play called 'Hamlet: Prince of Denmark') and the unique atom 'someone' which has been conjured from thin air.

The problem lies in expressing existential statements such as 'someone likes ice-cream' and so on. This is informally recast as there exists some person such that this person likes ice-cream. In first order predicate logic, we would formalise this as $\exists x \text{ likes}(x, \text{ice_cream})$. This can be turned into **likes(whatshisname,ice_cream)** (this is known as *Skolemisation*). Without going into technicalities, we give a legitimate context when this 'trick' can be done ---whenver we have no universal quantifiers (*i.e.* indicated by words such as all, everyone, etc) then we may introduce a unique atom (we should be able to guarantee its uniqueness) to stand for the 'someone'.

8. **mortal(X):- human(X).**

This is an example of a universally quantified statement. It is equivalent to $\forall x \text{ human}(x) \Leftrightarrow \text{mortal}(x)$.

Note that, in the **Prolog** version, this 'universal quantification' is implicit.

9. **pays_taxes(X):- person(X), rich(X).**

Again, the universal quantification is implicit in the **Prolog** version.

Here, we have a body with a conjunction of two goals. This could be avoided with **pays_taxes(X):- rich_person(X)**. Which you prefer depends on the way other relevant information is to be used or, how it is provided.

10. **takes(bill,umbrella):- raining.**

This is a version where it is true that 'Bill' takes his umbrella whenever it is raining.

Note that in many of these examples, there is no mention of how the truth of various statements change with time.

11. **no_supper(X):- naughty(X).**

Here, we might have tried to write \Rightarrow **supper(X):- naughty(X)**. This is, however, illegal in **Prolog** but not for syntactic reasons.

Another way of doing this might be **eats_supper(X,false):- naughty(X)**. This allows for a more uniform treatment of both those who are 'naughty' and those who aren't.

12. **employs(firebrigade,X):- man(X), height(X,Y), more_than(Y,6.0).**

Again, we have gone for the representation 'most likely' to be useful.

We could hide much of this as **firebrigade_employs(X):- over_six_foot(X)**.

<http://homepages.inf.ed.ac.uk/pbrna/prologbook/book.html>