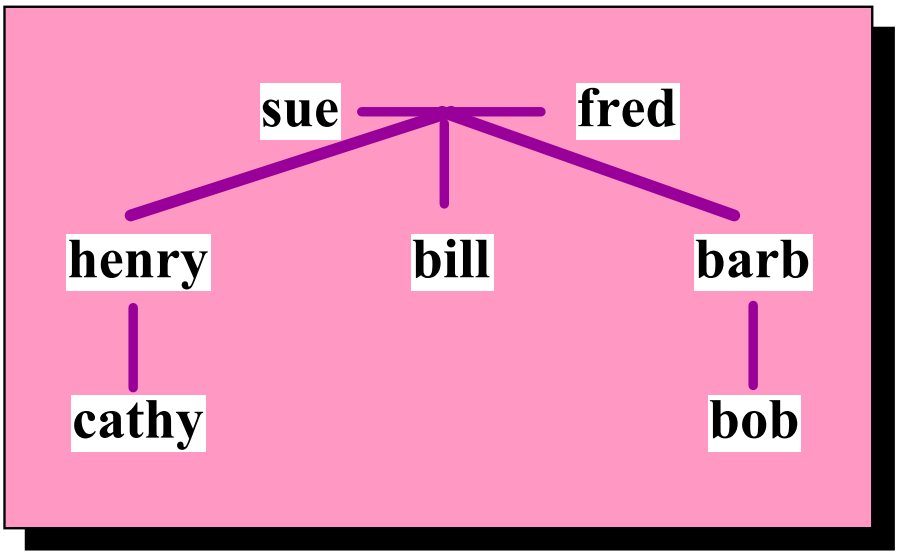
FAMILY TREE



(LOGIC PROGRAMMING)

This tutorial has been developed with reference to the following sources :-

"Prolog Programming for Artificial Intelligence" (3rd Edition) by I.Bratko, Addison-Wesley, 2001.

WHAT IS PROLOG?

One of the achievements of artificial intelligence research is that it is now possible to program a knowledge of predicate logic into a computer in the form of a computer language called Prolog.

Prolog enables you to provide facts and logical rules to the computer, in predicate logic notation; then allows you to ask questions, which the computer attempts to answer, based on the information you have given it.

Prolog is based on a simplified form of predicate logic. The predicates are represented in Prolog in the same way as in logic. However, because of keyboard limitations:

new syntax

- the AND symbol is a comma (**,**) instead of Λ;
- the OR (which is not used very often) is a semi-colon (**;**) instead of V; and
- the NOT is the symbol \+ instead of $\neg$.

Note:

- if you are not familiar with predicate logic we will be covering the topic very soon in lectures.

new syntax

Also, the IF-THEN rule (eg. IF A THEN B), which is represented in predicate logic as **A→B**, is represented in Prolog by the clause **B :- A** which says that **B** is true if **A** is true (think of the **:-** symbol as an arrow going backwards).

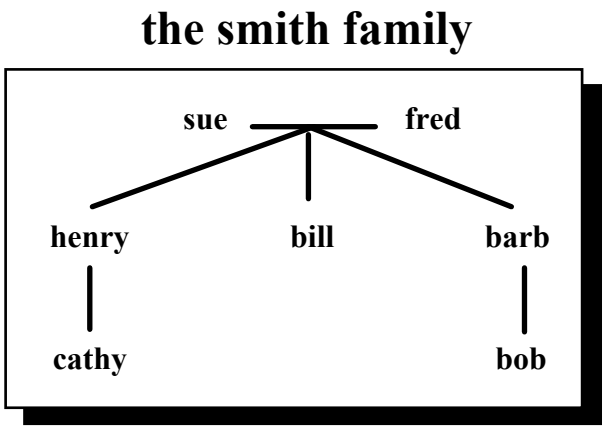
Quantifiers are not explicitly represented in Prolog. Instead, any clause which contains variables is regarded as being preceded by the appropriate universal quantifiers.

new syntax

Finally actual objects are represented by names (constants) which commence with a lower-case letter, while variables all commence with an upper-case letter.

A FAMILY TREE

In order to learn a little about Prolog, we will use it to provide information to the computer about a hypothetical family tree which looks like this:



We will also give it a set of rules defining family relationships, then question the knowledgebase we have developed to see if it can identify grandparents, aunts, cousins etc. of specific family members.

ADDING FACTS

We are going to add some basic facts for the Family Tree Knowledgebase. The first few facts tell the computer who is female and who is male. They define a particular attribute for each family member. The rest explain who is descended from whom within the family. Each of these facts identifies a relationship, between two family members. Notice that in Prolog, as in predicate logic, the relationship (denoted by the **predicate**) comes first, followed by the family members (denoted by the **arguments**).

The first facts are as follows:

new syntax

```
female(barb).          male(fred).
female(sue).           male(bob).
female(cathy).         male(henry).
                       male(bill).
```

The first female clause here states that Barb is female.

Note:

- that we can't specify names starting with uppercase as this would cause Prolog to treat them as variables (which we haven't learnt about yet) rather than constants;
- we don't have to predefine the people before we specify which gender they are; and
- we don't have to previously specify the concepts of male and female.

Specifying parent-child relationships between people is done as follows:

new syntax

```
parent(sue,  henry).       parent(fred, henry).
parent(sue, bill).         parent(fred, bill ).
parent(sue, barb).         parent(fred, barb).
```

The first parent clause here states that Sue is a parent of Henry.

Note:

- the order that we specify arguments (**sue** comes before **henry**).

We have not yet represented all the information that is contained in "the smith family" diagram. We need two more facts. One states that **henry** is a **parent** of **cathy** and the other that **barb** is a **parent** of **bob**.

1. Log on to the system and access **Prolog** as outlined in the appropriate **Getting Started with Prolog** information sheet.

2. Enter the text editor as outlined in the **Getting Started with Prolog** information sheet.

3. Determine for yourself what the last two missing facts should look like. Then type all of the required facts into the text editor. Don't forget the brackets, commas, and full stops and don't use any uppercase letters.

4. **Save** your work as "familytree.pl" as outlined in the **Getting Started with Prolog** information sheet.

5. Go back into Prolog and consult "familytree.pl" as outlined in the **Getting Started with Prolog** information sheet.

The Prolog command line allows us to interactively query the knowledgebase that we have created. For each question we type Prolog will initially only give us a **single response** to each question and then wait for further input. If you then type:

- a **semicolon** (**;**) and then press **enter**, it will give you **another response**, and then wait for input again;
- a letter **a** and then press **enter**, it will give you all possible responses; and
- just **enter** with no other input, it will stop evaluating.

Obtaining repeated responses in this way may sometimes result in Prolog producing the same response several times. It is doing this because it is searching out all possible reasoning paths which can lead to the answer required.

new syntax

6. To test the accuracy of the knowledgebase, type in the following queries and see if you get the correct response in each case. Some have multiple correct responses.

female(henry).	(ie. is henry a female?)
female(sue).	(ie. is sue a female?)
male(X).	(ie. who are all the males?)
female(X).	(ie. who are all the females?)
parent(X, henry).	(ie. who are henry's parents?)

Note:

- the use of **X** to represent a variable for which we wish Prolog to find a suitable value; and
- any word starting with a capital letter is considered to be a variable.

HOW DOES PROLOG WORK?

Prolog is a **pattern-matching** language. It takes the query that has been presented to it; then looks through the facts in the knowledgebase (from top to bottom) to see if any match the query. If it finds one that matches, ie. the predicate and argument(s) are the same, then it answers **yes**; if it doesn't, it answers **no**. For example, it will respond to the query **?- female(sue)** with the answer **yes** because this matches the third fact in the knowledgebase.

Things are slightly more complex if the query contains a variable. A variable is indicated by starting it with an upper-case letter. The query **?- male(X)** contains the variable **X**. Prolog will now look through its knowledgebase to find a fact that matches the predicate. When it finds the first such match it then sets the variable **X** to the value **fred** and reports that a match has been found, provided **X = fred**.

Having done this, if you ask it to find any more matches, by typing **semicolon (;)** and **enter**, it will then proceed on through the knowledgebase, and find another match, providing **X = bob** in this case. It is able to list all the males in the knowledgebase in this way.

The last query **?- parent(X, henry)** looks through the knowledgebase until a match is found to the predicate and to the second argument **henry**. When a partial match is found, it sets **X** to **sue** to produce a total match, and reports **X = sue**. When asked to find more matches it will then report **X = fred**. Prolog will answer **no** when no more matches can be found.

ADDING RULES

Now you need to give the computer some rules to use, so that it can identify family members with certain relationships to each other. The first rule is that any family member, **X**, is the **mother** of another member **Y**, if **X** is the **parent** of **Y** and **X** is **female**. It would look like this:

new syntax

mother(X, Y) :- parent(X,Y), female(X).

Note:

- the symbol **:-** is the Prolog symbol for IF and the comma between the two clauses is the Prolog symbol for AND.

8. Make up a similar rule to **mother** indicating that X is the father of Y if X is the parent of Y and X is male.

9. Also make up a rule that states that X is the grandparent of Y if X is the parent of another family member Z and Z is the parent of Y.

10. Type in the rules you have defined for the **mother**, **father**, and **grandparent** relationships. Check the rules very carefully to ensure brackets, commas, and full-stops are correct.

11. **Save** your work and consult "familytree.pl" in Prolog (you must use the consult command again to have your changes noticed by Prolog).

ASKING QUESTIONS

12. Type in the following queries and check the responses that you receive :-

mother(sue, henry).	(ie. is sue the mother of henry?)
grandparent(X, bob).	(ie. who are bob's grandparents?)
grandparent(sue, bill).	(ie. is sue a grandparent of bill?)
female(X), grandparent(fred, X).	(ie. is there a member of the family who is female and has fred as her grandparent)
parent(X,Y).	(ie. list all the parent-child pairs in the family)

To see how Prolog uses the rules to logically deduce new facts which are not explicitly in the knowledgebase, let us look at what happens when the following query was typed in:

grandparent(X, bob).

Follow this discussion carefully while referring to your printed listing.

Firstly, Prolog looked through the facts and found none with a predicate that matched this query. Then it looked through the rules and came across one which commenced with the predicate **grandparent**. It was able to match this by setting the variable **Y**, in the rule, to **bob**.

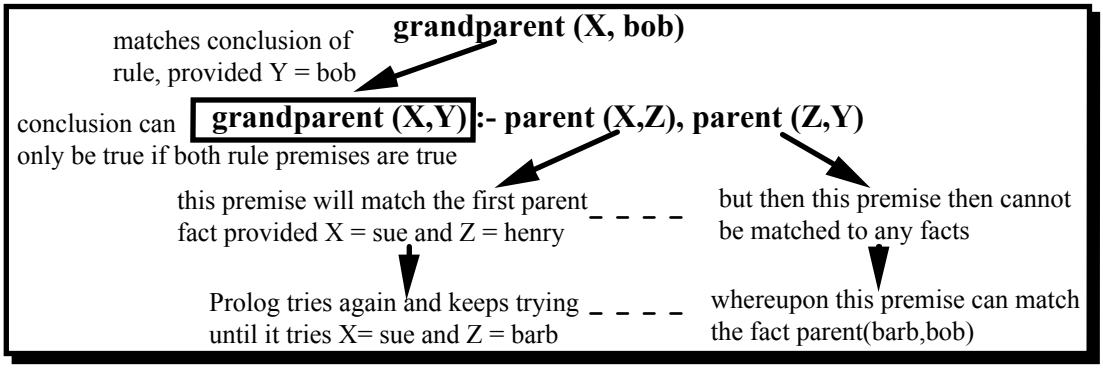
It can only regard this rule conclusion to be true if all the premises in the rule are true, so it must now ask two new queries:

parent(X, Z) and **parent (Z, bob)**

It tries the first query and can match it to the first **parent** fact in the knowledgebase, provided **X = sue** and **Z = henry**. Now it attempts to match the second query, with **Z** set to **henry**. Unfortunately there is no match since henry is not a parent of bob.

Prolog **backtracks** to the first rule premise and tries again. This time it matches to the second parent fact, with **X = fred** and **Z = henry** and again no match occurs because the second query **parent(henry,bob)** is false.

It keeps going like this until it sets **X = sue** and **Z = barb**. Now the second query can be matched by the last fact in the knowledgebase **parent(barb, bob)**. This means that with **X = sue**, **Z = barb**, and **Y = bob** the grandparent rule will fire, and the conclusion of the rule, **grandparent(sue, bob)**, becomes true. Prolog then indicates that it has found one grandparent of bob and asks you if you want it to find more.



PREDEFINED PREDICATES

As well as the predicates you define yourself, Prolog has a number of pre-defined predicates to make life a little easier. These don't always adopt the strict predicate-first convention.

One of the most important pre-defined predicates is the **is different from** predicate.

new syntax

X \== Y

This means that **X is different from Y**.

This predicate needs to be used if you wish to create a rule defining the sibling (ie children of the same parent) relationship. One could define the relationship thus:

sibling(X,Y) :- parent(Z,X), parent(Z,Y).

This says that X is a **sibling** of Y if some other family member Z is a **parent** of **both** X and Y. However, this rule, as it stands, allows family members to be their own siblings, which is not correct. To ensure that X and Y are different people, the \== predicate is used. The rule then becomes:

sibling(X,Y) :- parent(Z,X), parent(Z,Y), X \== Y.

If some of the terms specifying various family relationships are unfamiliar to you, you may access the **Dictionary of Strange Terms** from the **Resources** page on the website for an explanation.

FAMILY TREE QUESTION SHEET

STUDENT NAME .....

Question 1
Construct rules defining the sister and the brother relationships, using the previously given sibling relationship.

```
sister(X, Y) :- sibling(X, Y), female(X).
```

```
brother(X, Y) :- sibling(X, Y), male(X).
```

Type these, and the sibling rule, into the Family Tree Knowledgebase and then construct and enter appropriate queries to answer the following questions:

	query	answer
Does bill have a sister called barb?	`sister(barb, bill).`	.....
Does cathy have any siblings?	.....	.....
Who is henry's sister?	.....	.....
Who is henry's brother?	.....	.....

Question 2
Construct rules defining an uncle and an aunt relationship and give them here.

.....

.....

Type these into the Family Tree Knowledgebase and then construct and enter appropriate queries to answer the following questions:

	query	answer
Who is bob's uncle?	.....	.....
Is fred cathy's uncle?	.....	.....
Does cathy have an aunt?	.....	.....

Question 3
Construct a rule defining a cousin relationship and give it here.

.....

Type this into the Family Tree Knowledgebase and then construct and enter appropriate queries to answer the following questions:

	query	**answer**
Does bill have any cousins?	.....	.....
Who are cousins in the family?	.....	.....

Question 4
Type in the query: **?- parent(barb, X).** and note the answer.
Describe, in detail how Prolog would have arrived at that answer.

.....

.....

.....

Question 5
Type in the query: **?- father(X, henry).** and note the answer.
Describe, in detail how Prolog would have arrived at that answer.

.....

.....

.....

.....

.....

.....

.....

.....

.....

Question 6
Say **cathy** has just had a son **greg** and **bill** has just had a daughter **melanie**. What facts could be used to add this information to the knowledgebase?

.....

.....

Also define a new rule to establish the relationship **greatgrandparent**.

.....

Insert the facts above and the new rule into the knowledgebase. Then type in appropriate queries to ask the knowledgebase the following questions?

	query	**answer**
Does cathy have a parent?	.....	.....
Who are greg's great-grandparents?	.....	.....
Which family members have grandparents?	.....	.....

Note that Prolog will always answer No to a question if there is not sufficient information in the knowledgebase to prove it is Yes. This is called the **closed-world assumption**.

Question 7
Think up two new relationships of your own and define them in terms of existing relationships using appropriate rules. Note down the rules:

.....

.....

Insert these rules into the knowledgebase and test them out to see if they behave sensibly.