Getting started with SWI-Prolog

Download from the SWI Prolog webpage. SWI-Prolog provides an interactive command line environment. Type '`swipl`' on the command line to start Prolog.

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.3)


?-
```

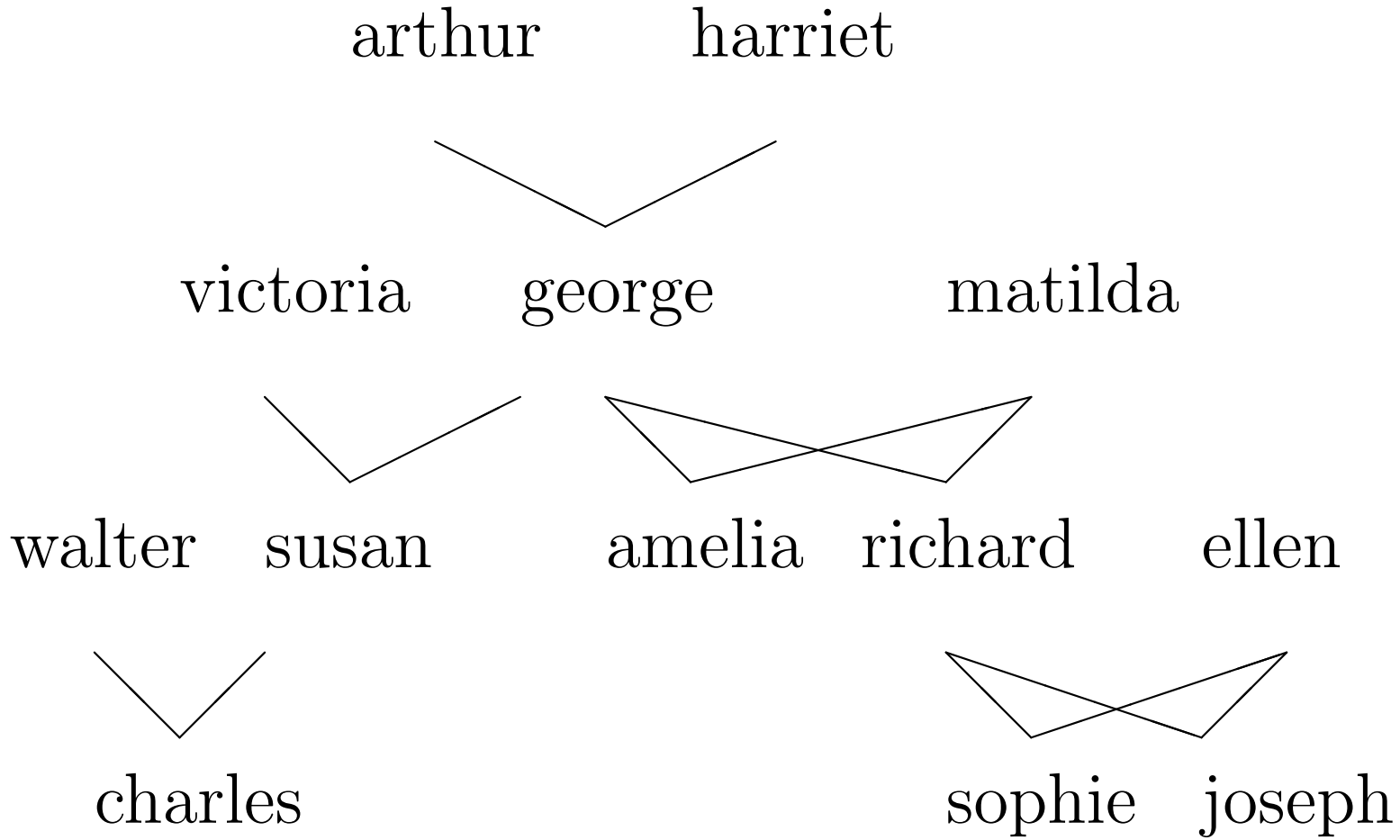
To consult a program file, type its name in square brackets.

```
?- [family].
```

Our program is compiled and the database can now be queried.

Type *Control-D* to leave Prolog when you are finished.

Family tree


arthur
harriet
victoria
george
matilda
walter
susan
amelia
richard
ellen
charles
sophie
joseph

Family database

The tree can be represented by the following facts in Prolog.

```
parent(arthur,george).
parent(harriet,george).
parent(george,richard).
parent(george,amelia).
...                          % 14 entries in total

male(arthur).
male(george).
...                          % 6 entries

female(harriet).
female(matilda).
...                          % 7 entries
```

We can query the database to find the parents of a particular individual, e.g.

```
?- parent(X,richard).

X = george ;

X = matilda
```

or, conversely, to find the children of an individual

```
?- parent(susan,Y).

Y = charles
```

Rules

We can extend the database to include a predicate `mother` by adding the following *rule* to the program file.

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

The rule gives a sufficient condition for the `mother` predicate to hold true. We can use the new predicate in queries, e.g.

```
?- mother(X,richard).


X = matilda
```

Prolog will search for a solution by searching for a solution to the equivalent composite query.

```
parent(X,richard), female(X).
```

The grandparent relation can be defined by the rule

```
grandparent(X,Y) :- parent(X,Z), parent(Z,Y).
```

Adding this rule to the program file, we can find the grandparents of an individual,

```
grandparent(X,richard).

X = arthur ;
X = harriet
```

or, conversely, find the grandchildren of an individual.

```
?- grandparent(george,Y).

Y = sophie ;
Y = joseph ;
Y = charles
```

The sibling relation can be defined by the rule

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

The not equals predicate '`\=`' enables us to discard solutions in which the value chosen for `X` is the same as that chosen for `Y` (no-one is a sibling of themselves).

Exercise 1. Define predicates for the other possible family relationships, in particular, try 'sister', 'aunt', 'cousin' and 'greatgrandparent'.

(You can make use of the predicates already given).

Disjunctive conditions

Suppose that the family database had been implemented differently with the `mother` and `father` predicates taken as primitive.

The `parent` predicate can be defined using two rules.

```
parent(X,Y):- father(X,Y).
parent(X,Y):- mother(X,Y).
```

These set out the two possible conditions under which the parent relation holds.

For `parent(X,Y)` to hold true, either `father(X,Y)` must hold true or `mother(X,Y)` must hold true.

Adding Recursion

The relation 'parent of' and 'grandparent of' are subsumed in the more general relation 'ancestor of'.

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(Z,Y), ancestor(X,Z).
```

These rules completely define the ancestor relation. They form a recursive definition because '`ancestor`' appears on both sides of the `:-` symbol.

The following won't work

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(Z,Y), ancestor(X,Z).


ancestor4(X,Y) :- ancestor4(X,Z), parent(Z,Y).
ancestor4(X,Y) :- parent(X,Y).
```

The program might be used, for example, to find the ancestors of an individual

```
?- ancestor(X,richard).

X = george ;
X = matilda ;
X = arthur ;
X = harriet
```

or, conversely, the descendants

```
?- ancestor(richard,X).

X = sophie ;
X = joseph
```