Practical Session 6: Prolog

1 Getting started

Prolog is a logic programming language. There are many implementations of Prolog available. We will use SWI-Prolog[1]

Logic programming has 3 kinds of statements: *facts*, *rules* and *queries*. a finir

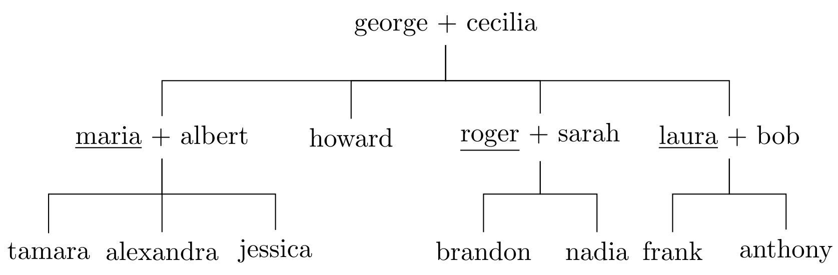
2 Facts

Prolog must be fed facts, which are statements which we know are true. They will be the basis of our programs. A fact start with a lowercase and end with a period :

```
prolog_is_simple.
life_is_beautiful.
likes(student, prolog).
cat(tom).
```

In the third line you see that facts can also have arguments: this particular fact tells us about the link between two terms.

Write facts which describe the following family tree in terms of `father/2` and `mother/2` and `male/1` and `female/1`. Names in bold indicate which member of the



[1]http://www.swi-prolog.org/

3 Queries

Once you have fed Prolog some facts, you can start asking it some queries. For instance :

```
?- likes(student, prolog).
true.
```

The ?- denotes the Prolog shell. In the shell, you can only ask queries. You cannot state new facts within the shell. Of course, you can do more than just querying facts as they were given explicitly. For instance, you could wish to know who likes Prolog. In this case, you whould use a variable, which is indicated by a capitalized first letter :

```
?- likes(Who, prolog).
Who = student.
```

In Prolog a comma , denotes a logical *and* whereas a semi-colon denotes a logical *or*. You can therefore query for all creatures who are cats or like prolog for instance:

```
?- cat(Who);likes(Who, prolog).
Who = tom ;
Who = student.
```

In this case there are two possible answers. You can use tab to obtain them successively. The `findall` predicate is also useful to obtain all possible answers:

```
?- findall(Creature, cat(Creature);likes(Creature,prolog),List).
List = [tom, student].
```

Write some queries to :

- know if George is the father of Tamara
- know if Anita is the mother of Brandon
- get all the children of Maria
- get all the sons of Roger
- check that Maria and Albert only have children together (use 2 `findall`'s)
- check that all fathers are male and all mothers are female (without checking this for every known person individually of course)

4 Rules

A *rule* allows us to use known facts to draw conclusions from our world. For example:

```
cat(tom).                 % this is a fact
animal(X) :- cat(X).      % this is a rule: if X is a cat, X is an animal
```

Would allow you to query for `?- animal(tom).` and obtain `true`.

Write rules to describe the following relationships:

- `parent(Parent,Child)`
- `son(Son,Parent)`
- `daughter(Daughter,Parent)`
- `grandfather(Grandfather,Grandchild)`
- `grandparent(Grandparent,Grandchild)`
- `brother(Brother,Sibling)`
- `sibling(Sibling1,Sibling2)`
- `havechildrentogether(Person1,Person2)`
- `uncle(Uncle,Person)`

5 Anonymous variables

The underscore character can be used to in place of variables when we are not interested in their value. This character will basically match anything, but each use will be considered a different variable, so you cannot do for instance:

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

Use the underscore wildcard to:

- determine whether Laura has one child
- add a `human/1` predicate so that *everyone* is a human. Can you also do this without the underscore?
- define a `isparent/1` predicate that checks whether a person as at least one child.

6 Recursive rules

Write a rule to describe to `ancestor` relationship. Hint: someone is an ancestor of someone if their are their direct parent, of if their are the direct parent of one of their ancestors.