Prolog – Two

August 20, 2003

Example (from one): let us consider a family of four individuals: John, Marry, Tom, and Celine. John is the father of Tom and Celine and Marry are mother of Tom and Celine. Tom is a boy and Celine is a girl. Tom is older than Celine.

To represent the fact that John is the father of Tom and Celine and Marry is the mother of Tom and Celine in Prolog, we write

$$\begin{gathered} father(john, tom). \\ father(john, celine). \\ mother(marry, tom). \\ mother(marry, celine). \end{gathered}$$

Here, *father* and *mother* are predicate symbols and *john, marry, tom,* and *celine* denote the four individuals of the family. The are *constants* – in Prolog, a string starts with a lower case letter denotes a constant. We edit a file called `p1.pl` and compile the file using Eclipse Prolog.

Now that we have the program. We can ask questions about the individuals and their relationships. To ask, whether *john* is a father of *tom*, we write in the query entry of Eclipse

$$father(john, tom).$$

Try the above and see what happens. Here is what you will see in the **Results** screen:

```
?- father(john, tom).
Yes (0.00s cpu)
```

To ask who is the father of *tom*, we write

$$father(X, tom).$$

Here, X is a *variable*. In Prolog, a string beginning with a upper case letter denotes a variable. Instead of the 'yes/no' answer we get the following:

```
?- father(X, tom).
X = john
Yes (0.00s cpu)
```

Notice the difference between answer for queries with variables and answer for queries without variable. Queries with variables are answered with an assignment for variable ($X = john$, meaning that $john$ is an answer for the query $father(X, tom)$. For queries without variables, the answer is 'yes/no'.

How do we represent the other facts?

$$\begin{gathered} male(tom). \\ female(celine). \\ older(tom, celine). \end{gathered}$$

We add the above to the program `p1.pl` and call the new program `p2.pl`. We compile the file and ask questions. You can try with your questions to see how Prolog answer them.

Let us figure out what happend if we ask the program the following questions:

- Is John a male? `?male(john).`   NO
- What is the gender of Celine? ?
- Are Celine and Tom siblings? `?sibling(celine, tom).` NO
- Who is a brother of Celine? `?brother(X, celine).` NO
- Who is a brother of Tom? `?brother(X, tom).` NO

What is wrong? The problem is that we have in our mind a lot of information, such as John – being a father – is normaly a male; or Celine and Tom – having the same parent – are siblings. However, this information is not represented in our program. For this reasons, the answer will be NO.

We can, of course, represent the fact that $John$ is a male and $Marry$ is a female using the facts:

$$male(john).$$
$$female(marry).$$

Asking now whether John is a male, we will get the correct answer. Observe that to record the gender of some one, denoted by x, we need either $male(x)$ or $female(x)$. This solution will not be good if we have a large number of individuals. **Can we do it better than that?** A reasonable assumption would be that every individual is either a male or a female. So, we can choose to represent the 'female' individuals and write some rules to deduce that someone who is not a female is a male. This can be written as follows:

$$male(X) :- not\ female(X).$$

The *not* in the above rule is called the *negation-as-failure* operator. The rule is read as if *female(X)* cannot be proven then *male(X)* is true.

Suppose that we have the program `p3.pl` as follows:

```
father(john, tom).
father(john, celine).
mother(marry, tom).
mother(marry, celine).
female(celine).
female(marry).
male(X) :- not female(X).
older(tom, celine).
```

Now, if we ask our program about the gender of every individual, we will get the correct answer. As you can see, a Prolog program can be extended incrementally by adding new atoms (facts) and rules. Now, suppose that we wanted to add to `p3.pl` also some information about the addresses of the four individuals. John and Marry are in Las Cruces whereas their two children are studying in California, San Francisco, to be precise. We could do so by adding the facts:

```
livein(john, las_cruces).
livein(marry, las_cruces).
livein(tom, san_francisco).
livein(celine, san_francisco).
```

As you can see, constants can also contain the underscore symbol. Let `p4.pl` be the program consisting of `p3.pl` and the above four atoms. Let's compile it and ask queries about the gender of the individuals in the story, we get the same result. We also can ask queries about who is living where. The program will also return the correct answers. Now, let us ask the query

```
? male(las_cruces).
```

We get the answer ‘YES’. This is certainly not what we wanted but why it is so? The culprit is the rule that defines whoever is not a female is a male! Of course, the error is ours. For Prolog, a string like ‘john’ is not much different than the string ’las_cruces’ except that the former is shorter and they contain different characters. How will you solve this problem? This is the first homework for this class. The second homework asks you to extend the program so that we can answer the questions about relationship between members of an extended family.

Homework 1: Extending the program `p4.pl` with necessary rules and facts so that the following queries will be answered correctly:

- Questions about gender of people?
- Questions about who is living where?
- Questions about the relationships between Tom and Celine.

Homework 2: Let the program resulting from HW 1 be `p5.pl`. Extending the program with the following facts:

- John has a brother Paul.
- Marry has a brother Ringo and a sister Ono.
- Paul is living in San Francisco.
- Everyone lives in San Francisco or Las Cruces.

Define the relationship *uncle* and *aunt*. Your program should be able to answer queries about the relationship between the new individuals and the previously mentioned ones. It should also giving correct answer about the living place of each individual. Furthermore, the answers should not be changed if we add some facts about tables and chairs in the class.