**Lab 2 -Prolog Syntax and Error Messages**

**Terms, Facts, Queries, and Rules**

The basic constructs in Prolog are: facts, rules and queries. These are all built out of Terms.

**Terms**

There are four kinds of Terms in Prolog:

1. integer A positive or negative numbers, e.g. 23,4500, -360
2. atom There are three different classes of atoms:

(1) All text constants (text constants can contain letters, numbers and underscores) beginning with a lowercase letter, e.g. george, j987\_abe;

(2) All terms enclosed in a single quote 'Bill Gates';

(3) Certain special symbols: +, -, ...

1. variable Begins with an uppercase letter or underscore \_, for example George, or \_ron. Note, the underscore alone \_ is an anonymous variable and two occurrences of \_ are different (i.e. they cannot be bound together).
2. Complex Term also called structures are built out of a functor followed by a sequence of arguments. The arguments are put in ordinary parentheses, separated by commas and placed after the functor. The functor must be an atom (i.e., a variable cannot be used as a functor). In other words, complex terms have the form functor(argument1, ..., arguementN).

For example, likes(mary,john) is a complex term with functor=likes, argument1=mary, and argument2=john. Other examples of complex terms are:

smart(john), loves(dt2284,ai), kisses(mary, X), likes(Y,icecream), parent(X,Y), append(X,Y,Z).

Note: arguments can also be complex terms!

**Facts**

Facts are the simplest form of Prolog construct consisting of a predicate and a list of arguments. The syntax for a fact is:

pred(arg1, arg2, ... argN).

where

1. pred Is the name of the predicate
2. arg1, ...argN - arguments can be any legal Prolog term.
3. N Is the arity or number of arguments of the predicate. A predicate of arity 0 is simply pred
4. . The syntactic end of all Prolog clauses

**Queries**

Once we have some facts in our Prolog knowledge base, we can query the Prolog interpreter to find out whether or not a fact is true given the current knowledge base. A fact is true if it is listed in the knowledge base, (or if it is the head of a rule and all the facts in the body of the rule can be proved true).

Prolog queries work by pattern matching. The query pattern is called a goal. If there is a fact that matches the goal, then the query succeeds and Prolog responds with yes or true. If there is no matching fact, then the query fails and Prolog responds with no.

Prolog's pattern matching is called unification. In the case where the logicbase contains only facts, unification succeeds if the following three conditions hold:

1. The predicate named in the goal and logicbase are the same.
2. Both predicates have the same arity.
3. All of the arguments are the same

Let’s assume we have the following knowledge base:

room(kitchen).

room(office).

room(hall).

room('dining room').

room(cellar).

The first query we will look at asks if there is an office room. To pose this, we would enter that goal followed by a period at the listener prompt.

?- room(office).

yes

Prolog will respond with a 'yes' if a match was found. If we wanted to know if the attic was a room, we would enter that goal.

?- room(attic).

no

**Rules**

There is an important difference between facts and rules. A fact like parent(tom, liz). is always true. Rules specify things that are true if some condition is satisfied. Therefore rules have: (1) a condition part (the right-had side of the rule or body), and (2) a conclusion part (the left hand side of the rule or head). The syntax for defining a rule is:

head :- body.

where

1. head is a predicate definition (just like a fact)
2. :- is the neck symbol, sometimes read as "if"
3. body is one or more goals (a query)
4. . is the syntactic end of all Prolog clauses

If all of the goals in the body can be proved then the head of the rule is taken to be true.

**Prolog Connectives**

The Prolog connectives include:

1. As we discussed above, in Prolog the neck symbol :- defines the IF THEN connective. For example, the Prolog rule

happy(john):-eating(john). can be understood as specifying that john is happy IF john is eating.

1. In Prolog the comma symbol , defines the AND connective. For example the Prolog rule

happy(john):-eating(john), watchingtv(john).

can be understood as specifying that john is happy if john is eating AND john is watching tv.

1. In Prolog the semi-colon symbol ; defines the OR connective. For example For example the Prolog rule

happy(john):-eating(john),watchingtv(john);playingbaskteball(john).

can be understood as specifying that john is happy if (john is eating and john is watching tv) OR john is playing basketball. You can also define an OR relation by having two separate rules handling each side of the OR. For example, the rule

happy(john):-eating(john),watchingtv(john);playingbaskteball(john).

could be rewritten as two rules

happy(john):-eating(john),watchingtv(john).

and

happy(john):-playingbaskteball(john).

## Writing your own knowledge bases

There are two basic steps to writing a Prolog knowledge base:

1. Start a text editor and open a new file called Name.pl (where Name is a name of your choice), write your knowledge base, and then save the file in your working directory
2. You can then load and query this knowledge base in the same way as you loaded and queried the previous knowledge bases. When making changes to Name.pl, you have to save the file again and you have to load it into Prolog again.

As an exercise, write a Prolog knowledge base called kb4.pl specifying the following facts and rules:

1. Harry is a wizard.
2. Hagrid scares Dudley.
3. All wizards are magical.
4. Uncle Vernon hates anyone who is magical.
5. Aunt Petunia hates anyone who is magical or scares Dudley.

Try writing the solution yourself before looking at the hints below.

**Hints:**

1. **Harry is a wizard.** This sentence assigns a property (being a wizard) to a specific individual (Harry). It hence states a fact. Properties are usually encoded as Prolog predicates, and specific individuals as atoms. Remember that atoms cannot start with a capital letter; i.e., Harry is not an atom, while harry and 'Harry' are atoms.
2. **Hagrid scares Dudley**. This sentence says that a particular relation (scare) holds between two specific individuals (Hagrid and Dudley). Hence, it states a fact.
3. **All wizards are magical**. This sentence does not talk about a specific individual, but about all individuals with a certain property (all individuals with the property of being a wizard). It expresses a general rule about wizards. The sentence could be reformulated as if somebody is a wizard, then he/she is magical or as for all X, if X is a wizard, then X is magical.
4. **Uncle Vernon hates anyone who is magical**. This sentence states that a particular relation (hate) holds between a specific individual (Uncle Vernon) and any other individual with the property of being magical. You need a rule to express this. The sentence could be reformulated as for all X, if X is magical, then Uncle Vernon hates X.
5. **Aunt Petunia hates anyone who is magical or scares Dudley**. This sentence is very similar to the above. It states that Aunt Petunia hates anyone who is magical and in addition, she hates anyone who scares Dudley. This can be reformulated as follows: for all X, if X is magical or scares Dudley, then Aunt Petunia hates X, which is the same as saying for all X, if X is magical, then Aunt Petunia hates X, and for all X, if X scares Dudley, then Aunt Petunia hates X. That is, we express the disjunction by specifying two rules: one for each disjunct.

**Solution** – kb4.pl

Load this database into Prolog. Prolog should give you a short answer saying that it compiled your knowledge base (or just show you the prompt). If it gives you a different answer, there might be a problem with your knowledge base. Skip ahead to the next section to see which are the things that Prolog may complain about.

Ask the following queries:

1. Does Aunt Petunia hate Hagrid? (The answer should be yes.)
   * hates(aunt\_petunia,hagrid).
2. Who does Uncle Vernon hate? (The answer should be Harry.)
   * hates(uncle\_vernon,X).
3. Who does Aunt Petunia hate? (The answer should be Harry and Hagrid. Type semicolon to get the second answer).
   * hates(aunt\_petunia,X).

**Interpreting the error messages and warnings that Prolog may give you**

**Syntax Errors**

Prolog will tell you if the things that you are writing are syntactically incorrect. Download the knowledge base [**kb\_syntax\_error.pl**](file:///C:\Users\SVETLA~1.HEN\AppData\Local\Temp\7zO4A6D9DA0\syntax_error\kb_syntax_error.pro) to your working directory and load it into Prolog. You will get something like the following answer.

?- Warning: (c:/.../prolog/workspace/kb\_syntax\_error.pl:2):Redefined static procedure wizard/1

ERROR: c:/.../prolog/workspace/kb\_syntax\_error.pl:5:0: Syntax error: Operator expected

ERROR: c:/.../prolog/workspace/kb\_syntax\_error.pl:11:0: Syntax error: Operator expected

ERROR: c:/.../prolog/workspace/kb\_syntax\_error.pl:33:0: Syntax error: Unexpected end of clause

ERROR: c:/.../prolog/workspace/kb\_syntax\_error.pl:44:0: Syntax error: Unexpected end of file

% c:/.../Prolog/Workspace/kb\_syntax\_error.pl compiled 0.02 sec, 9,996 bytes

This tells you that there are four syntax errors. It also tells you that the Prolog interpreter thinks that the errors are in lines 6, 12, 15, and 18. And it gives you some indications as to what might cause the problems (operator expected, unexpected end of clause/file). Which are the mistakes? (Solution at the end of the lab-sheet)

**Singleton Variables**

Another thing that you might get complaints about are singleton variables; i.e., variables which are used only once in a clause. Download the knowledge base [**kb\_singleton\_vars.pl**](file:///C:\Users\SVETLA~1.HEN\AppData\Local\Temp\7zO4A6D9DA0\singleton_variables\kb_singleton_vars.pro) and load it into Prolog. You will get something like the following:

% c:/.../Prolog/Workspace/kb\_singleton\_vars.pl compiled 0.02 sec, 1,292 bytes

?- Warning: (c:/.../prolog/workspace/kb\_singleton\_vars.pl:8):Singleton variables: [X, Y]

% c:/.../Prolog/Workspace/kb\_singleton\_vars.pl compiled 0.00 sec, 0 bytes

Prolog tells you that X and Y are singleton variables in the clause starting at line 8. Prolog warns you when it detects singleton variables, because singleton variables are often due to a typo (as in our example). Sometimes you do want to use variables which are mentioned only once in a clause. To avoid getting the singleton variable warning, you should use the anonymous variable \_ instead of a named variable in this case.

**Clauses not together**

Finally, Prolog complains when different clause belonging to the definition of the same predicate are not together in the file. Here is an example: [**kb\_clauses\_not\_together.pl**](file:///C:\Users\SVETLA~1.HEN\AppData\Local\Temp\7zO4A6D9DA0\clauses_not_together\kb_clauses_not_together.pro). When you load this knowledge base Prolog answers:

?- Warning: (c:/.../prolog/workspace/kb\_clauses\_not\_together.pl:14):Clauses of hate/2 are not together in the source-file

% c:/.../Prolog/Workspace/kb\_clauses\_not\_together.pl compiled 0.01 sec, 2,224 bytes

The first clause of the definition of the predicate hate/2 is separated from the other two clauses by a rule with the head magical(X)

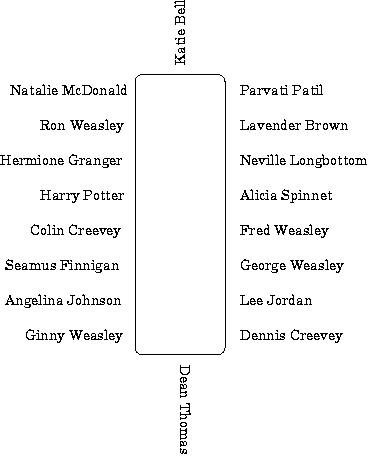
**Exercises**

**Exercise 1:**

Modify the wizard KB we’ve seen previously so that X is a wizard if X's father or mother is a wizard?

**Exercise 2: The Gryffindor table**

The following picture shows who is sitting at the Gryffindor table. Define the functor sits\_right\_of/2 to represent who is sitting right of whom. sits\_right\_of(X,Y) should be true if X is to the right of Y.



**Solution**

Based on this knowledge base, formulate the rules defining the following predicates:

sits\_left\_of/2: sits\_left\_of(X,Y)should be true if X is to the left of Y.

(Hint: For all X and all Y, X is sitting left of Y, if Y is sitting right of X.)

are\_neighbors\_of/3: are\_neighbors\_of(X,Y,Z) should be true if X is to the left of Z and Y is to the right of Z.

next\_to\_each\_other/2: next\_to\_each\_other(X,Y) should be true if X is next to Y.

(Hint: X and Y are next to each other, if X is right of Y or if X is left of Y.)

Test your implementation by asking queries. For example:

1. *Is Lavender to the right of Parvati?*
   * sits\_right\_of(lavender,parvati).
2. *Is Lavender to the right of Neville?*
   * sits\_right\_of(lavender,neville).
3. *Who is to the right of Hermione?*
   * sits\_right\_of(X,hermione).
4. *Who is sitting at the table?*
   * *sits\_right\_of(X,Y).*
5. *Who is sitting two seats to the right of Hermione?*
   * are\_neighbours\_of(hermione,Y,harry).
6. *Who is sitting between Neville and Fred?*
   * are\_neighbors\_of(fred,Neville,Z).

(Partial solution – griff1.pl)

**Solution to Syntax Errors:**

1. Line 5 is: Scare(hagrid,dudley). Functors always have to be atoms, but atoms cannot start with a capital letter. Correction: scare(hagrid,dudley).
2. Line 11: hate(uncle vernon,X) :- magical(X). Atoms cannot contain spaces (and neither can variables, by the way) unless the whole atom is enclosed in single quotes. So, the problem is uncle vernon. Correction: hate(uncle\_vernon,X) :- magical(X). or hate('uncle vernon',X) :- magical(X).
3. Line 14: hate(aunt\_petunia,X :- magical(X). The head of the rule (what comes before :-) should be a complex term. All arguments have to be between an opening and a closing parenthesis. In Line 14, the closing parenthesis is missing. Correction: hate(aunt\_petunia,X) :- magical(X).
4. Line 17: hate(aunt\_petunia,X) :- scare(X,dudley) All clauses (facts and rules) have to end with a full stop. In Line 17, this full stop is missing. Correction: hate(aunt\_petunia,X) :- scare(X,dudley)