Lab 5 – 08226 Artificial Intelligence

This lab tutorial will introduce you to backtracking.

1.0 Backtracking

Start SWI-Prolog-Editor from the Windows Menu system and create a new Prolog file called **Lab5.pl** and store it in **G:/08226/Lab 5/**.

When Prolog finds a sub-goal of a rule that fails it does not simply give up. Prolog will backtrack to previous sub-goals within the rule and try other instances of results in the program. If it finds another result then it will continue to move forward once more evaluating sub-goals within the rule.

Consider the following program:

```
food(apple,fruit).      /* apple is a fruit */

food(tomato,fruit).     /* tomato is a fruit */

food(lettuce,salad).    /* lettuce is a salad */

food(beef,meat).        /* beef is a meat */

food(cucumber,salad).   /* cucumber is a salad */


display_salad_food :- food(Food,salad),
                      write(Food), write(' is a salad'), nl.
```

We are going to ask Prolog to display what salad foods it has.

```
?- display_salad_food.
```

Even though we have two salad foods in the program database, our rule will only display the first of them, namely that **lettuce is a salad**.

Let's go through exactly what Prolog does with our rule.

The **display_salad_food** rule has the first sub-goal of the **food** fact. Prolog looks from the top of the program database and finds the **food(apple,fruit)** fact. However this fails the first sub-goal because we are looking for salad. Instead of quitting, Prolog then tries to execute the sub-goal to the left of the current sub-goal. We are currently on the first sub-goal so it executes this sub-goal again. Prolog then looks at the next **food** fact in the program database and finds the **food(tomato,fruit)** fact. The same thing happens, so then Prolog looks for the next **food** fact in the program database which is **food(lettuce,salad)** fact. The first sub-goal will now complete and so Prolog will move to the second sub-goal, which is the display the food is a salad to the query window followed by the final sub-goal which is to move the cursor down to the next line.

The rule then finishes, which finishes our program. So as you can see a rule does not automatically keep going to see if there are any more results after it has found one, like the query window does.

2.0 Fail

The predicate **fail**, when encountered, always fails which forces backtracking. Therefore we can force backtracking when we want to.

The predicate **fail** is used predominantly for listing data from a number of facts.

Consider the following program:

```
food(apple,fruit).      /* apple is a fruit */
food(tomato,fruit).     /* tomato is a fruit */
food(lettuce,salad).    /* lettuce is a salad */
food(beef,meat).        /* beef is a meat */
food(cucumber,salad).   /* cucumber is a salad */

display_salad_food :- food(Food,salad),
                      write(Food), write(' is a salad'), nl, fail.
display_salad_food.
```

This code is almost identical to the previous code. The difference is the added **fail** predicate to the rule, and the extra fact below the rule. We are going to ask Prolog to display what salad foods it has.

```
?- display_salad_food.
```

Our rule will now display all the salads, namely that **lettuce is a salad** and **cucumber is a salad**.

Let's go through exactly what Prolog does with our rule.

The **display_salad_food** rule has the first sub-goal of the **food** fact. Prolog looks from the top of the program database and finds the **food(apple,fruit)** fact. However this fails the first sub-goal because we are looking for salad. Instead of quitting, Prolog then tries to execute the sub-goal to the left of the current sub-goal. We are currently on the first sub-goal so it executes this sub-goal again. Prolog then looks at the next **food** fact in the program database and finds the **food(tomato,fruit)** fact. The same thing happens, so then Prolog looks for the next **food** fact in the program database which is **food(lettuce,salad)** fact. The first sub-goal will now complete and instantiate **Food** to **lettuce**, and so Prolog will move to the second sub-goal, which is the display the food is a salad to the query window followed by the next sub-goal which is to move the cursor down to the next line. The final sub-goal **fail** then fails causing backtracking. The sub-goal **nl** fails when backtracking, so does the sub-goals with the predicate **write**, so Prolog ends up back at the first sub-goal where it left a marker for the instantiated **lettuce**.

This sub-goal then looks at the next **food** fact in the program database and finds the **food(beef,meat)** fact. However this fails the first sub-goal because we are looking for salad. Prolog then looks at the next **food** fact in the program database and finds the **food(cucumber,salad)** fact. The first sub-goal will now complete and instantiate **Food** to **cucumber**, and so Prolog will move to the second sub-goal, which is the display the food is a salad to the query window followed by the next sub-goal which is to move the cursor down to the next line. The final sub-goal **fail** then fails causing backtracking. The sub-goal **nl** fails when backtracking, so does the sub-goals with the predicate **write**, so Prolog ends up back at the first sub-goal where it left a marker for the instantiated **cucumber**. This sub-goal then looks for the next **food** fact in the program database but cannot find one so the rule finishes. Due to the rule failing the fact that we added by the same name is found and succeeds. It is normal to add this fact so that our program succeeds and thus continues, otherwise the program may finish.

2.1 Fail Exercise

Consider the following program:

```
hello_world :- write('Hello '), fail, write('World'), nl.

hello_world.
```

What do you expect the program to output to the query window?

Check your answer here: **Fail Exercise Answer**

3.0 Repeat

We can use the predicate **repeat** to force a sub-goal to succeed.

Consider the following program:

```
hello_world :- repeat, write('Hello '), write('World'), nl, fail.

hello_world.
```

If we called this rule, the first sub-goal **repeat** will always succeed, and so Prolog then moves to the two **write** sub-goals which displays **Hello World** to the query window. Prolog then moves to the **nl** sub-goal which moves the cursor to the next line. The final sub-goal **fail** then fails causing backtracking. The sub-goal **nl** fails when backtracking, so does the sub-goals with the predicate **write**, so Prolog ends up back at the first sub-goal **repeat** which always succeeds. Therefore Prolog then moves to the two **write** sub-goals which displays **Hello World** to the query window. Prolog then moves to the **nl** sub-goal which moves the cursor to the next line. The final sub-goal **fail** then fails causing backtracking. And so on forever and ever and ever and ever and .....

Therefore the predicate **repeat** can be used as a bookmark when backtracking is used.

3.1 Repeat Exercise 1

Consider the following program:

```
hello_world :- write('Hello '), repeat, write('World'), nl, fail.

hello_world.
```

What do you expect the program to output to the query window?

Check your answer here: **Repeat Exercise Answer 1**

3.2 Repeat Exercise 2

Consider the following program:

```
hello_world :- repeat, write('Hello '), repeat, write('World'),
               nl, fail.

hello_world.
```

What do you expect the program to output to the query window?

Check your answer here: **Repeat Exercise Answer 2**

4.0 Fail Exercise Answer

Consider the following program:

```
hello_world :- write('Hello '), fail, write('World'), nl.

hello_world.
```

What do you expect the program to output to the query window?

Answer:

Hello will be output to the query window and then the rule fails and so finishes.

Return to your exercise here: **Fail Exercise**

5.0 Repeat Exercise Answer 1

Consider the following program:

```
hello_world :- write('Hello '), repeat, write('World'), nl, fail.

hello_world.
```

What do you expect the program to output to the query window?

Answer:

First **Hello World** will be output to the query window. Then the word **World** will be output to the query window forever, e.g.

```
Hello World

World

World

World

World

World
```

Return to your exercise here: **Repeat Exercise 1**

6.0 Repeat Exercise Answer 2

Consider the following program:

```
hello_world :- repeat, write('Hello '), repeat, write('World'),
               nl, fail.

hello_world.
```

What do you expect the program to output to the query window?

Answer:

First **Hello World** will be output to the query window. Then the word **World** will be output to the query window forever, e.g.

```
Hello World
World
World
World
World
World
```

Return to your exercise here: **Repeat Exercise 2**