For Office Use Only

# Artificial Intelligence – Assignment 2

**Question 1 –** the 'maximum' predicate

```
maximum([X], X).
maximum([H|T], H) :- maximum(T, Y), H >= Y.
maximum([H|T], X) :- maximum(T, X), X > H.
```

The way this predicate works is by defining a **boundary case** of just one list element that says the highest value of that list is the one value in it – this is used later on so that the recursive predicates can come to an end and iterate back through the goals. (Stack Exchange Inc., 2014)

The first **general case** works with the idea that the head of the current list IS the maximum value, and then goes on to check the maximum within the tail of the list isn't greater than the current max.

The second **general case** takes the only other case to consider – that the head of the list ISN'T the maximum. It then goes on to check for the maximum value within the tail of this list and make sure that it IS in fact greater than the current head.

**Question 2 –** the 'duplicate_nth' predicate

```
duplicate_nth(1, [H|T], [H,H|T]).
duplicate_nth(N, [H|T], [H|T2]) :-
     N > 1,
     N1 is N - 1,
     duplicate_nth(N1, T, T2).
```

The way I designed this predicate to work is that it recursively negates 1 from the value of N until the head of the sub-list it's checking is the value we want to duplicate (i.e. N = 1), before doing so.

The **boundary case** then works on the idea that when N is equal to 1 then the head of the list is the value we want to duplicate, so the sub-list [H|T] becomes the sub-list [H,H|T], where everything's the same but the head is there twice.

The **general case** is for when N is greater than 1, which indicates that the head of the current list isn't the value that we want to duplicate. When this is the case, the head for both of the lists must be the same, but the tail of the second list will be different (hence '[H|T] ' and [H|T2]') – it then attempts to prove the predicate again, but this time using the tails of the lists and an N value decremented by 1. This repeats until N is 1 and the boundary case is met.

**Question 3 –** semantic network inheritance

```
instance(city, london).
instance(city, lincoln).
instance(town, boston).

subclass(settlement, city).
subclass(settlement, town).
subclass(building, cathedral).
subclass(building, townhall).

has(city, cathedral).
has(city, townhall).
has(town, townhall).

has(X, Y) :- instance(C, X), has(C, Y).
has(X, Y) :- instance(C, X), has(C, D), subclass(Y, D).
```

For this task I decided to just have simple predicates defining the relationship between values rather than anything more complex. They're formatted as such:

- `instance(X, Y)` – Y is an instance of X
- `subclass(X, Y)` – Y is a subclass of X
- `has(X, Y)` – X has a Y

The only thing that gets more complicated is the 'has' predicate, which I defined in two ways – that if something isn't explicitly written to *have* something, then it checks that whatever X is an instance of (C) has instead. The second one does the exact same thing, but then defines X as 'having' the superclass of anything its own base type has.

With this, the assignment task is met, with the addition of the following two lines at the end of the code to show the difference in the queries (see the next page):

```
instance(cathedral, stmarys).          % stmarys is a cathedral
has(lincoln, stmarys).                 % stmarys belongs to lincoln
```

**Query 1 –** before St. Mary's was included:

```
?- has(lincoln, Y).

Y = cathedral ;
Y = townhall ;
Y = building ;
Y = building ;
false.
```

**Query 2 –** after St. Mary's was included:

```
?- has(lincoln, Y).

Y = cathedral ;
Y = townhall ;
Y = building ;
Y = building ;
Y = stmarys.
```

The difference here then is that, after St Mary's was defined as an instance of a cathedral and that it belongs to Lincoln, querying for all of Lincoln's buildings now includes St. Mary's on the list and doesn't return false.

The main "issue" with the way I've designed this is that there is no clear way to state that the cathedral Lincoln owns as a result of being a city is *not* separate from the cathedral St. Mary's that it's explicitly stated to own. As a result, for any counting purposes this predicate becomes useless (especially since it doesn't differentiate between the building types and the superclass of 'building'), but for simple queries such as "does Lincoln have a Cathedral" or "does Lincoln have a building" it works just fine.

This meets the example queries given, in that it defines Boston as having a town hall and Lincoln as having at least one building – what it doesn't do however is allow us to ask if a city or a town has a building, due to stack overflow issues I was encountering whilst trying to write that in.

## References

Stack Exchange Inc. (2014)
***prolog – Two clause definition to find the maximum number in a list – Stack Overflow*** [online]
Available from: http://stackoverflow.com/questions/1816057/two-clause-definition-to-find-the-maximum-number-on-a-list
[Accessed 18th March 2014]