CT331 Assignment 3

Declarative Programming with Prolog

Andrew Reid East
16280042

https://github.com/reideast/ct331_assignment3

Question 1

1.	sunny AND warm	True
2.	sunny AND cold	False
3.	sunny OR cold	True
4.	(sunny OR cold) AND warm	True
5.	happy XOR sunny	False
6.	warm XOR (NOT happy)	True
7.	early NAND happy	False
8.	(late NOR (NOT early)) AND (windy OR (NOT warm))	True
9.	(cloudy AND windy) AND (warm AND early)	False
10.	(cloudy AND windy) XOR (warm OR early)	True

Question 2

2.1

```
teaches(INSTRUCTOR, STUDENT) :- instructs(INSTRUCTOR, CLASS),
                                takes(STUDENT, CLASS).
```

2.2

```
12 ?- teaches(bob, X).
X = tom ;
X = mary ;
X = joe.
```

2.3

```
13 ?- teaches(X, mary).
X = bob ;
X = ann.
```

2.4

False

Prolog begins matching `teaches(ann, joe).` line-by-line through the database. It will not match anything until it finds the line which defines the rule `teaches`, which has two un-unified variables, `INSTRUCTOR` and `STUDENT`, which it unifies to `ann` and `joe`. It then examines the remainder of the rule, which begins with the relation `instructs`. One argument to `instructs` is already bound, but the variable `CLASS` is not, and it remains ununified. The antecedent proceeds with a conjunction to a

relation, `takes`. All the arguments to `takes` are already unified. It encounters the full stop character, and has finished processing this line.

Prolog will then proceed to backtrack through the database, attempting to match lines that will satisfy the unbound variables. Looking for `instructs(ann, CLASS)`, it finds the fact `instructs(ann, ct345)`, and binds `ct345` to CLASS. It must then backtrack again to find a fact that makes the entire AND clause true. With all the variables unified, it is looking for the precise fact: `takes(joe, ct345)`. It will loop through the database, but will not find this fact. Therefore, it will undo its last unification, `ct345` to `class`. Then, it continues looping to look for new information that matches `instructs(ann, CLASS)`. It reaches the end of the database without finding anything, and because of negation-as-failure, the query is false.

2.5

```
classmates(A, B) :- takes(A, CLASS), takes(B, CLASS).
```

```
2 ?- classmates(mary, joe).
true ;
false.

3 ?- classmates(tom, mary).
true ;
true.

4 ?- classmates(mary, tom).
true ;
true.

5 ?- classmates(joe, bob).
false.

6 ?- classmates(bob, joe).
false.
```

Question 3

3.1

```
8 ?- [H|T] = [1,2,3].
H = 1,
T = [2, 3].
```

3.2

```
12 ?- [HEAD | [TAIL_HEAD | TAIL_TAIL]] = [1,2,3,4,5].
HEAD = 1,
TAIL_HEAD = 2,
TAIL_TAIL = [3, 4, 5].
```

3.3

```
contains1(ELEM, [ELEM | _]).
```

3.4

```
contains2(LIST, [_ | LIST]).
```

3.5

```
24 ?- contains1(X, [1,2,3]).
X = 1.
```

Question 4

```
% Base case - element is the head of a list
contains(ELEM, [ELEM | _]).

% Reduction case - list is reduced to everything except the head of the list
% Element is in the list IF element is in the tail of the list
contains(ELEM, [_ | TAIL]) :- contains(ELEM, TAIL).

```