Joseph Agass (jag208)
Hippolyte Le Carreres (hca102)

MAS assignment 1 (Group 172)

Exercise 1

boiling(water). % water is used as a constant as it is assumed that this specific water is boiling.

drinking_tea(ravi, alexandra). % Ravi and Alexandra are of type constant since these people specifically are drinking tea together.

likes(alexandra, rooibos). % Alexandra likes a specfic type of tea, a constant called Rooibos

has_motherboard(X) :- isComputer(X). % Every computer X has to have a motherboard. Variable X was used as there could be different computers that have a motherboard.

has_motherboard(X) :- isSmartphone(X). % Every smartphone X has to have a motherboard. Variable X was used as there could be different smartphones that have a motherboard.

functions(X) :- hasBattery(X),

isPluggedIn(X). % A laptop X only functions if it has battery and it is plugged in.

can_play_video_game(X) :- hasGamingDevice(X), masAssignment(X). % A person X can play a video game if they have a gaming device and they have completed their MAS assignment.

game_is_good(X) :- engaging(X), immersive(X). % A video game X is good if it is engaging and immersive.

Exercise 2

1) **apple = orange.**

Does not unify since they are different constants and therefore different atoms.

2) **cat(X) = cat(fluffy).**

Does unify since fluffy will be unified with the variable X.

Joseph Agass (jag208)
Hippolyte Le Carreres (hca102)

3) **tree(oak, X) = tree(oak, pine).**

Does unify as X is unified with pine.

4) **apple = Orange.**

Does unify since Orange is a variable and therefore unified with apple.

5) **hobby(reading, X) = hobby(X, running).**

Is not unified as X can't be instantiated as both reading and running. Variables cannot be unified with two different atoms at once.

6) **house(roof, walls) = house(roof, walls).**

Does unify as terms on either side are identical.

7) **likes(alice, bob) = likes(alice, charlie).**

Does not unify, as charlie and bob are different atoms and alice can't be unified to both at the same time.

8) **vehicle(car, wheels(4)) = vehicle(car, wheels(X)).**

Does unify as the variable X is unified with 4.

9) **f(a, f(b), c) = f(a, f(b)).**

Does not unify as the terms have different arity.

10) **f(f(A), b) = f(f(B), C).**

Does unify as the variable A is unified with the variable B and the constant b is unified with the variable C. B and b are not the same term.

11) **sum(3, X) = sum(Y, 5).**

Joseph Agass (jag208)
Hippolyte Le Carreres (hca102)

Does unify such that X is unified with 5 and Y is unified with 3.

Exercise 3

1) t(r(r(c)),c).

This succeeds since r(r(c)) and c are different, which means that therule t(Q,R) :- q(Q),r(R),not(Q=R). holds. Moreover, the knowledge base contains r(_) which can take on any value which results in this rule being true.

2) t(r(c),r(c)).

This fails since r(c) is identical to r(c) as it is the same thing and therefore the rule t(Q,R) :- q(Q),r(R),not(Q=R). does not hold. This is because this rule would only be true if as stated q(Q) is not equal to r(R).

3) t(a,a).

This is infinite since, after failing to satisfy the first clause because a = a, Prolog then gets stuck on the second clause in an infinite loop, trying to satisfy t(a, a) by repeatedly calling the same clause.

4) t(A,B).

While t(A, B) is successful since it fulfills the first clause, Prolog enters an endless loop in the second clause for values like A = a or A = b, repeatedly trying to satisfy t(A, B) through recursive calls without a base case.

Exercise 4

6) By moving the \= term to the end of the rule definition, the rule works as it should and provides the possible moves for the horizontal (and also vertical in the second clause) line of the rook.

9)

Defining the predicate reachable/6 with recursion is not good as it runs into various problems. For instance, recursion would lead the predicate to be repeatedly called until the target is reached. Moreover, it could also get stuck on moving back to the original square, which would result in an infinite loop, as the piece would move back and forth between the square it went to after moving from its base square and then back to its base square.

Joseph Agass (jag208)
Hippolyte Le Carreres (hca102)

10)

The only piece that would allow recursions for this rule is the pawn. This is because the pawn is the only piece that cannot go back to its starting square as it will always move one square forward. However, all the other pieces can go back to their starting square, which could lead to them getting stuck in an endless loop as mentioned in the previous answer.