**Q 3. Unification:** Indicate which of the following pairs of terms can be unified together? If

they can't be unified, please provide the reason for it. In case of error, indicate the error. If

they can be unified successfully, wherever relevant, provide the variable instantiations that

lead to successful unification. (Note that `=' indicates unification)

1. food(bread, X) = Food(Y, soup)

**NO**: This won’t work as prolog is case sensitive (food vs Food, Food is a variable, can’t be a complex term)

2. Bread = soup

**Yes:** first letter being a capital letter means “Bread” is a variable which can unify with anything

soup = Bread

3. Bread = Soup

**Yes:** first letter being a capital letter means “Bread” is a variable which can unify with anything

Soup = Bread

4. food(bread, X, milk) = food(Y, salad, X)

**No**: X can’t both be salad and milk

5. manager(X) = Y

**Yes:** Y = manager(X)

6. meal(healthyFood(bread), drink(milk)) = meal(X,Y)

**Yes**: X = healthyFood(bread) Y = drink(milk)

7. meal(eat(Z), drink(milk)) = [X]

**No:** a list can’t unify with a functor that doesn’t use lists

8. [eat(Z), drink(milk)] = [X, Y | Z]

**Yes :** Here X and Y are variables and Z is the tail. Because there are two variables in the left hand side, Z = []. Because of that X = eat([]) and Y = drink(milk)

9. f(X, t(b, c)) = f(l, t(Z, c))

**Yes :** X = l and Z = b (it can be both sides)

10. ancestor(french(jean), B) = ancestor(A, scottish(joe))

**Yes:** similar to last question, A will unify french(jean) and B will unify with scottish(joe)

11. meal(healthyFood(bread), Y) = meal(X, drink(water))

**Yes:** same thing, X = healthyFood(bread) and Y = drink(water)

12. [H|T] = [a, b, c]

**Yes:** H = a T = [b,c].

13. [H, T] = [a, b, c]

**No:** H and T aren’t the head and tail here, they’re simply variables. However one list has 3 arity, the other has 2 therefore unification can’t happen

14. breakfast(healthyFood(bread), egg, milk) = breakfast(healthyFood(Y), Y, Z)

**No:** because of the first argument Y has to be bread. However, the second argument says that Y = egg. Therefore, there is contradiction and no unification

15. dinner(X, Y, Time) = dinner(jack, cook( egg, oil), Evening)

**Yes:** X = jack Y = cook(egg,oil) Time = Evening

16. k(s(g), Y) = k(X, t(k))

**Yes:** X = s(g) Y = t(k)

17. equation(Z, f(x, 17, M), L\*M, 17) = equation(C, f(D, D, y), C, E)

**No:**  the second argument D tries to unify with 17 and with x which is a contradiction and therefore can’t unify.

18. a(X, b(c, d), [H|T]) = a(X, b(c, X), b)**No:** list [H|T] can’t find a list to unify with (b isn’t a list)

**Q 4. Queries**

Determine the type of each of the following queries (ground/non-ground), and explain what will Prolog respond for each of these queries (write all the steps of unications and resolutions for each query)?

**THE ANSWERS ALWAYS ASSUME USER INPUTS “ ; “ AS LONG AS POSSIBLE**

1. ? field(hit\_transfer,engineering).

**Ground query**

**Prolog will return true.**

It won’t find the fact field(hit\_transfer, engineering) directly

Instead, prolog will use the rule: field(X, Y) :- course(X, Z), field(Z, Y).

X = hit\_transfer and Y = engineering

It will look for course(hit\_transfer, Z) and find that Z = mechanical

Then, it will look if field(Z, Y) is valid.

field(Z,Y) is field(mechanical, engineering) which is defined in the database

**true; false.**

2. ? lab\_number(fine\_arts,X).

**Non-ground query**

**X = 10**

Prolog will try to find an X which is valid. In the database there is lab\_number(fine\_arts,10) so X will unify with 10

If we keep going it will try this rule lab\_number(X, Z) :- course(X, Y), lab\_number(Y, Z).

But won’t find anything because there is no course(fine\_arts, \_) in the database

**X = 10; false.**

3. ? field(computer, literature).

**Ground query**

**false.**

Prolog will try to find this fact in the database, won’t find it, then try the rule

field(X, Y) :- course(X, Z), field(Z, Y).

However it won’t find a course(computer, \_).

Therefore the result is

**false.**

4. ? course(X,Y).

**Non-ground query**

**X = hit\_transfer, Y = mechanical ; X = web\_design, Y = computer ;**

**X = design\_methods, Y = fine-arts ; X = poetry, Y = literature ;**

**X = leadership, Y = management ; X = biology, Y = medicin.**

Prolog will find every possible value for X and Y in the database (basically return every valid course statement that has 2 arity)

5. ? student(adrian).

**Ground query**

**true; true; false.**

Prolog will look for a student rule with 1 arity. This is the rule:

student(X):- student(X,\_).

Therefore, it will look for any student that has “adrian" as the first argument and since there is student(adrian, web\_design) in the database it will return **true**.

Then, it will try the rule

student(X, Y) :- field(Z, Y), student(X, Z).

where X is adrian and Y is \_.

It will find that Z = web\_design and that Y = computer as there is the expression course(web\_design,computer) in the database.

As this rule is valid, it will return **true** once more.

Finally, nothing else can be found so **false** is returned

6. ? student(anna, engineering).

**Ground query**

**true; false.**

Prolog will look whether this fact exists but won’t find it. It will try the rule

student(X, Y) :- field(Z, Y), student(X, Z).

X = anna, Y = engineering, Z = hit\_transfer

It will look for field (hit\_transfer, engineering) using this rule

field(X, Y) :- course(X, Z), field(Z, Y).

X = hit\_transfer, Y = engineering Z = mechanical  
Since there is a fact course(hit\_transfer, mechanical) and there is a fact field(mechanical, engineering) the original rule will prove valid

This is valid and there is no other possibility, therefore the query returns

**true; false.**

7. ? student(X, engineering).

**Non-ground query**

**anna; daniel;** **adrian; false.**

Prolog won’t find any facts that directly unify X with a student instead it must go with the rule

student(X, Y) :- field(Z, Y), student(X, Z).

Y = engineering, then prolog will try X = anna so Z = hit\_transfer as the fact student(anna, hit\_transfer) is defined in the database

For this to be valid we need field(hit\_transfer, engineering) to be true. Prolog will use the same logic as in question 6  
(proof:  
field(X, Y) :- course(X, Z), field(Z, Y).

X = hit\_transfer, Y = engineering Z = mechanical  
Since there is a fact course(hit\_transfer, mechanical) and there is a fact field(mechanical, engineering) the original rule will prove valid)

so **anna** is a valid answer.

Then prolog returns to the original rule and tries with another student as X. This time X = daniel. If X = daniel then using the original rule Z must be hit\_transfer as student(daniel, hit\_transfer) is a fact in the database. This is the same idea that we just proved above in the same question. field(hit\_transfer, engineering) must be true for this to work. Therefore **daniel** is a valid answer.

After, prolog tries with student adrian. We have the fact student(adrian, web\_design) so Z = web\_design in the original rule.   
Prolog will see if field(web\_design, engineering) is valid. As it does not exist as a fact, it will again use the rule

field(X, Y) :- course(X, Z), field(Z, Y).

X = web\_design Y = engineering Z = computer (

Since we have fact course(web\_design, computer) and have fact field(computer, engineering) the original rule is valid. Therefore, **adrian** is a valid answer.

It will look for the other students but it will be invalid. Therefore, the answer is **anna; daniel; adrian; false.**

8. ? student(X,fine-arts), course(fine\_arts, Y).

**Non-ground query**

**false.**

There is no fact or rule that allows student(X, fine-arts) and there is no also fact or rule that allows course(fine\_arts, Y) (because fine-arts is always the second argument of course facts and on top of that the facts are written with a “-“ and not “\_” between fine and arts as in the query) therefore prolog returns

**false.**

9. ? field(\_,X).

**Non-ground query**

**X = engineering ; X = engineering ; X = art ; X = social ; X = buisiness ; X = engineering ; X = engineering ;**

**X = art ; X = social ; X = buisiness ; false.**

First, prolog will search for every fact with field that have 2 arguments and unify X with the second argument. This will unify X with each of these “engineering, engineering, art, social, business” such that prolog returns **X = engineering ; X = engineering ; X = art ; X = social ; X = buisiness ;** as long as we input the “**;**”

Then prolog will use the rule:  
field(X, Y) :- course(X, Z), field(Z, Y).

And prolog will find every possible value for Y using the course facts as long as we input the “;”

1. course(hit\_transfer, mechanical)

X = hit\_transfer, Z = mechanical, there is a field such that field(mechanical, engineering) therefore Y = engineering. Prolog will then return **X = engineering**

1. course(web\_design, computer)

X = web\_design, Z = computer there is a field such that field(computer, engineering) therefore Y = engineering. Prolog will return again **X = engineering**

1. course(design\_methods, fine-arts)

X = design\_methods, Z = fine-arts there is a field such that field(fine-arts, art) therefore Y = art. Prolog will return **X = art**

1. course(poetry, literature).

X = poetry, Z = literature there is a field such that field(literature, social) therefore Y = social. Prolog will return **X = social**

1. course(leadership, management)

X = leadership, Z = management there is a field such that field(management, buisiness) therefore Y = engineering. Prolog will return **X = buisiness**

Prolog won’t find any other valid possibilities so if we input “;” again it will return **false;**

10. ? lab\_number(\_,X), field(X,Y).

**Non-ground**

**false.**

Since the second argument of lab\_number must always be a number and there is no number in field arguments, prolog won’t find any valid possibility and return **false.**

11. ? lab\_number(X,15), field(X,Y).

**Non-ground**

**X = mechanical, Y = engineering ;**

**X = hit\_transfer, Y = engineering ;**

**false.**

First prolog will look at the fact with lab\_number that has a 15 in the second argument.

That is: lab\_argument(mechanical, 15). Therefore, **X = mechanical.**

It will look at which Y makes field(mechanical, Y) valid.

**Y = engineering.**

It will try to find other field values using the field rule but won’t find any. Then, it will go to the lab\_number rule

lab\_number(X, Z) :- course(X, Y), lab\_number(Y, Z).

since Z = 15 and Y = mechanical, X = hit\_transfer since there is the fact course(hit\_transfer, mechanical)

Therefore, lab\_number(hit\_transfer, 15) is true

Thus for lab\_number(X,15), field(X,Y),

X = hit\_transfer and then Y = engineering as we proved previously using the field rule since field(hit\_transfer, engineering) is valid.

(proof:  
field(X, Y) :- course(X, Z), field(Z, Y).

X = hit\_transfer, Y = engineering Z = mechanical  
Since there is a fact course(hit\_transfer, mechanical) and there is a fact field(mechanical, engineering) field(hit\_transfer, engineering) is valid)

So prolog will return

**X = hit\_transfer, Y = engineering**

12. ? student(X), !, student(X,\_). % note to cut here

**Non-ground query**

**anna; anna; false.**

First, student(X) invokes the rule student(X):- student(X,\_).

This will then go to the first student fact it can find

student(anna, hit\_transfer).

Then the cut operator. No backtracking from here, X is committed and must be anna

Prolog will test whether student(anna) can be valid with the second part of the query after the !: (student(X,\_)), it will see there’s no problem and will return **X = anna**.

However prolog will not end there.  
While it won’t reset to the first part of the query (to the left of the !), it will reset (backtrack) the right part of the query after the cut operator: student(X,\_). It will try to find other possibilities where X can be anna.  
There is 1 other possibility: student(X,\_) where X is still anna and \_ doesn’t matter (but we proved earlier that \_ would be engineering in question 6)

As such, prolog will return **X = anna** again.

(Earlier proof for the second case of anna:

student(X, Y) :- field(Z, Y), student(X, Z).

X = anna, Y = engineering, Z = hit\_transfer

It will look for field (hit\_transfer, engineering) using this rule

field(X, Y) :- course(X, Z), field(Z, Y).

X = hit\_transfer, Y = engineering Z = mechanical  
Since there is a fact course(hit\_transfer, mechanical) and there is a fact field(mechanical, engineering) the original rule will prove valid)

In summary: prolog returns **anna; anna; false.**

13. ? student(X), student(X,\_), !.

**Non-ground query**

**anna**

The process is similar here, student(X) will use the 1 argument student rule to unify with X = anna, it will then validate with student(X,\_) without any problem and return **anna**

However, this time the cut operator is at the end preventing any backtracking or reset with what’s on the left of the cut operator. Thus, the query stops there since there’s nothing after the cut operator.

14. ? course(X,\_), \+ student(\_,X). % \+ is for negation (not)

**Non-ground query**

**biology**

Prolog will go through every course and look if there’s any student that studies that course. For example, it find the fact course(hit\_transfer, mechanical) and look for any student that studies hit\_transfer. It will find that anna does so this is not valid (the not operator means we are looking for a course that NO student studies).

Prolog will do the same for web\_design (adrian studies it), design\_methods (ava studies it), poetry (jack studies it), leadership (lee studies it).   
It will then try to find a student that studies biology. However, no fact in the database shows a student that studies biology.   
  
Thus, it will go for the rule

student(X, Y) :- field(Z, Y), student(X, Z).

Where Y has to be biology.

If Y is biology, then we are looking for a field that has biology as a second argument.

There is no such fact in the database, therefore we must use the field rule

field(X, Y) :- course(X, Z), field(Z, Y).

Where Y has to be biology. Automatically we see that this is impossible as it still requires biology as a second argument. We can still look deeper though and see that no course is in the database such that course(X,Z) and field(Z,biology) are valid.

Therefore, the original statement is invalid and biology is not studied by any student.

Prolog will return  
**X = biology**