COMP 348: Principles of Programming Languages
Assignment no.1

Duc Nguyen - Antonio Verdicchio - Zahra Nikbakht

Gina Cody School of Computer Science and Software Engineering
Concordia University, Montreal, QC, Canada

Summer 2020

Contents

1 Example to demonstrate LaTeX

Here is an example for the latex file

```
all_sections(CNAM, CNUM, L) :-
findall(L, takes_course(_, CNAM, CNUM, L), X),
sort(X, L).
```

2 Question 4

2.1 Modified Query

```
team(X), member(S, X),
|    findall([A,B], takes_course(S, A, B, _), L1),
|    list_to_set(L1, L),
|    length(L, NN),
|    write(S), write('has taken'), write(NN),
|    write(' course '), nl, fail.
```

3 Question 6:

Indicate which of the following pairs of terms can be unied together. If they cant be unied, please provide the reason for it. In case of error, indicate the error. If they can be unied successfully, wherever relevant, provide the variable instantiations that lead to successful unication. (Note that : indicates unication)

3.1 food(bread, X) = Food(Y, soup)

In this case, unification will not be successful since **food** is a functor and **Food** is not. For this unification to be successful, both sides would have to contain matching functors such as **food**.

3.2 Bread = soup

Since we have a variable and an atom, the variable will be instantiated to the value of the atom, thus the terms unify.

3.3 Bread = Soup

Since we have two variables, the variable will be instantiated to the value of the other variable or vice versa, thus the terms unify.

3.4 food(bread, X, milk) = food(Y, salad, X)

In this case the functors match, but the arguments do not. Y will be instantiated to 'bread', but X will be instantiated to either 'salad' or 'milk', meaning it cannot match with both of the atoms. This means the terms will not unify. If there

were three variables, lets say X,Y,Z instead of X,Y,Y, then the terms would unify.

3.5 manager(X) = Y

In this case unification will be successful and **Y** will be instantiated to **manager(X)**.

3.6 meal(healthyFood(bread), drink(milk)) = meal(X,Y)

In this case unification will be successful with **X** instantiated to **healthyFood(bread)** and **Y** instantiated to **drink(milk)**.

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

In this case, unification is not successful as the arities and functors do not match.

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

In this case unification is successful with **X** being instantiated to **eat([ ])**, **Y** being instantiated to **drink(milk)**, and **Z** being instantiated to [ ].

3.9 f(X, t(b, c)) = f(l, t(Z, c))

In this case unification is successful with **X** being instantiated to **1**, and and **Z** being instantiated to **b**.

3.10 ancestor(french(jean), B) = ancestor(A, scottish(joe))

In this case unification is successful with **A** being instantiated to **french(jean)**, and and **B** being instantiated to **scottish(joe)**.

3.11 meal(healthyFood(bread), Y) = meal(X, drink(water))

In this case unification is successful with **X** being instantiated to **healthyFood(bread)**, and and **Y** being instantiated to **drink(water)**.

3.12 [H|T] = [a, b, c]

In this case unification is successful with **H** being instantiated to **a**, and **T** being instantiated to **[b, c]**.

3.13 [H, T] = [a, b, c]

In this case unification is not successful as the number of elements in the left list does not equal the number of elements in the right list. The list on the left would have to be composed of 3 variables for unification to be successful.

3.14 breakfast(healthyFood(bread), egg, milk) = breakfast (healthyFood(Y), Y, Z)

In this case unification is not successful as **Y** cannot be instantiated to both **bread** and **egg** at the same time.

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

In this case unification is successful with **X** being instantiated to **jack**, **Y** being instantiated to **cook(egg, oil)**, and **Time** being instantiated to **Evening**.

3.16 k(s(g), Y) = k(X, t(k))

In this case unification is successful with **X** being instantiated to **s(g)**, and **Y** being instantiated to **t(k)**.

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

In this case, unification is not successful as the term **f(x, 17, M)** does not unify with the term **f(D, D, y)**. These two terms do not unify as **D** cannot be instantiated to both **x** and **17** at the same time.

3.18 a(X, b(c, d), [H|T]) = a(X, b(c, X), b)

In this case unification is not successful since **[H|T]** cannot unify with **b** since **b** is an atom.

4 Question 8

4.1 ? magic(Hermione)

In this non ground query, we will find all atoms such that they satisfy magic(Hermione). First, magic(Hermione) is unified with the head of the rule magic(X):- house_elf(X). So X is instantiated to Hermione. Through resolution, we go to the body of the rule and house_elf(Hermione) is now our goal to satisfy and it is treated as a new non ground query. We have house_elf(dobby) in our knowledge base so house_elf(Hermione) unifies with house_elf(dobby) and Hermione is instantiated to dobby so dobby satisfies our goal and thus it is one answer to the initial query.

There are no more answers for house_elf(Hermione), so using backtracking, we reach the next rule for magic(X). magic(Hermione) is unified with the head of the rule magic(X):- wizard(X). So X is instantiated to Hermione. Then in the resolution step, wizard(Hermione) is now our goal to satisfy. We have wizard(dobby) in our knowledge base so wizard_elf(Hermione) unifies with wizard(dobby) and Hermione is instantiated to dobby so dobby is another answer of the query.

There are no more answers for wizard(Hermione), so using backtracking, we reach the next rule for magic(X). magic(Hermione) is unified with the head of the rule magic(X):- witch(X). X is instantiated to Hermione. Then in the resolution step, we make a transition to the body of the rule and witch(Hermione) is now our goal to satisfy. We have witch(hermione), witch(mcGonagall) and witch(rita_skeeter) in our knowledge base so Hermione is first instantiated hermione and when we continue the search, there are other answers mcGonagall and rita_skeeter to the non groud query witch(Hermione). Through backtracking, we can see that there is no other answer for our initial query.
The final answer to the query is:

```
Hermione = dobby;
Hermione = dobby;
Hermione = hermione;
Hermione = mcGonagall;
Hermione = rita_skeeter;
```

4.2 ? magic(hermione)

This is a ground query, so we expect either true or false as an answer. First, Prolog searches the database from top to bottom. It matches the query with head of the rule magic(X):- house_elf(X). So X is instantiated to hermione. Now our goal is house_elf(hermione) (through resolution) and we treat it as a new ground query. We can see that house_elf(hermione) cannot match any clause in our database so we go to the next rule through backtracking: magic(X):- wizard(X). Now X is instantiated to hermione. In the resolution step, we make a transition to the body of the rule and our goal is to satisfy (find) wizard(hermione). We can clearly see in the database that hermione is a wizard so the result of our

goal query (?- wizard(hermione).) is true. Therefore we can tell that the initial query also returns true.

5 Question 11

The prolog file is in the file *question-11.pl* Run query:

```
lucas(N, X)
```

to call the rule and execute with:

- N: the desired index of the lucas sequence.
- X: the return list.