

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	3
2	Question 4	3
2.1	Modified Query	3
3	Question 6:	3
3.1	food(bread, X) = Food(Y, soup)	3
3.2	Bread = soup	3
3.3	Bread = Soup	3
3.4	food(bread, X, milk) = food(Y, salad, X)	3
3.5	manager(X) = Y	4
3.6	meal(healthyFood(bread), drink(milk)) = meal(X, Y)	4
3.7	meal(eat(Z), drink(milk)) = [X]	4
3.8	[eat(Z), drink(milk)] = [X, Y Z]	4
3.9	f(X, t(b, c)) = f(l, t(Z, c))	4
3.10	ancestor(french(jean), B) = ancestor(A, scottish(joe))	4
3.11	meal(healthyFood(bread), Y) = meal(X, drink(water))	4
3.12	[H T] = [a, b, c]	4
3.13	[H, T] = [a, b, c]	4
3.14	breakfast(healthyFood(bread), egg, milk) = breakfast (healthy- Food(Y), Y, Z)	5
3.15	dinner(X, Y, Time) = dinner(jack, cook(egg, oil), Evening)	5
3.16	k(s(g), Y) = k(X, t(k))	5
3.17	equation(Z, f(x, 17, M), L*M, 17) = equation(C, f(D, D, y), C, E)	5
3.18	a(X, b(c, d), [H T]) = a(X, b(c, X), b)	5
4	Question 7	6
4.1	field(hit_transfer, engineering)	6
4.2	? lab_number(fine_arts,X).	6
4.3	? field(computer, literature).	6
4.4	? course(X,Y).	7
4.5	? student(adrian).	7
4.6	? student(anna, engineering).	8
4.7	? student(X, engineering).	9
4.8	? student(X,fine-arts), course(fine_arts, Y).	9
4.9	? field(_,X).	10
4.10	? lab_number(_,X), field(X,Y).	11
4.11	? lab_number(X,15), field(X,Y).	12
4.12	? student(X), !, student(X,_).	12
4.13	? student(X), student(X,_), !.	13
4.14	? course(X,_), \+ student(_,X).	14
5	Question 8	16
5.1	? magic(Hermione)	16
5.2	? magic(hermione)	16
6	Question 11	17

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 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)

3.1 $\text{food}(\text{bread}, X) = \text{Food}(Y, \text{soup})$

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 $\text{Bread} = \text{soup}$

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 $\text{Bread} = \text{Soup}$

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 $\text{food}(\text{bread}, X, \text{milk}) = \text{food}(Y, \text{salad}, X)$

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 $\text{manager}(\mathbf{X}) = \mathbf{Y}$

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

3.6 $\text{meal}(\text{healthyFood}(\text{bread}), \text{drink}(\text{milk})) = \text{meal}(\mathbf{X}, \mathbf{Y})$

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

3.7 $\text{meal}(\text{eat}(\mathbf{Z}), \text{drink}(\text{milk})) = [\mathbf{X}]$

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

3.8 $[\text{eat}(\mathbf{Z}), \text{drink}(\text{milk})] = [\mathbf{X}, \mathbf{Y} \mid \mathbf{Z}]$

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

3.9 $\text{f}(\mathbf{X}, \text{t}(\mathbf{b}, \mathbf{c})) = \text{f}(\mathbf{1}, \text{t}(\mathbf{Z}, \mathbf{c}))$

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

3.10 $\text{ancestor}(\text{french}(\text{jean}), \mathbf{B}) = \text{ancestor}(\mathbf{A}, \text{scottish}(\text{joe}))$

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

3.11 $\text{meal}(\text{healthyFood}(\text{bread}), \mathbf{Y}) = \text{meal}(\mathbf{X}, \text{drink}(\text{water}))$

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

3.12 $[\mathbf{H} \mid \mathbf{T}] = [\mathbf{a}, \mathbf{b}, \mathbf{c}]$

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

3.13 $[\mathbf{H}, \mathbf{T}] = [\mathbf{a}, \mathbf{b}, \mathbf{c}]$

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 $\text{breakfast}(\text{healthyFood}(\text{bread}), \text{egg}, \text{milk}) = \text{breakfast}(\text{healthyFood}(\text{Y}), \text{Y}, \text{Z})$

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

3.15 $\text{dinner}(\text{X}, \text{Y}, \text{Time}) = \text{dinner}(\text{jack}, \text{cook}(\text{egg}, \text{oil}), \text{Evening})$

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 $\text{k}(\text{s}(\text{g}), \text{Y}) = \text{k}(\text{X}, \text{t}(\text{k}))$

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

3.17 $\text{equation}(\text{Z}, \text{f}(\text{x}, 17, \text{M}), \text{L} * \text{M}, 17) = \text{equation}(\text{C}, \text{f}(\text{D}, \text{D}, \text{y}), \text{C}, \text{E})$

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 $\text{a}(\text{X}, \text{b}(\text{c}, \text{d}), [\text{H}|\text{T}]) = \text{a}(\text{X}, \text{b}(\text{c}, \text{X}), \text{b})$

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

4 Question 7

4.1 field(hit_transfer, engineering)

Query type: ground.

Output: true

Steps:

- **Unify** the query with the head of rule:

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

- **Instantiate** X to hit_transfer, Y to engineering.

- **Resolve** to two new queries:

```
course(hit\_transfer, Z), field(Z, engineering).
```

- Both queries are evaluated.
- Both goals prove true \rightarrow **succeeds**. Thus, the answer to the query is true.

4.2 ? lab_number(fine_arts,X).

Query type: non-ground.

Output:

```
X = 10;  
false.
```

Steps:

- **Unify** with the head of fact:

```
lab_number(fine_arts,10).
```

- **Instantiate** X to 10.

4.3 ? field(computer, literature).

Query type: ground.

Output: false.

Steps:

- **Unify** the query with the head of the rule:

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

- **Instantiate** *computer* to X , *literature* to Y .
- **Resolve** to two new queries:

```
course(computer, Z), field(Z, literature).
```

- Both queries are evaluated. Once the first goal succeeds, try the next one on the right with the same instantiation.
- After trying all possible cases, no cases prove true. The query is not successful.

4.4 ? course(X,Y).

Query type: non-ground.

Output:

```
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.
```

Steps:

- **Unify** Unify with each statement of course() with 2 arities.
- **Instantiate:**
 - X is instantiated as hit_transfer, Y is instantiated as mechanical
 - X is instantiated as web_design, Y is instantiated as computer
 - X is instantiated as design_methods, Y is instantiated as fine-arts
 - X is instantiated as poetry, Y is instantiated as literature
 - X is instantiated as leadership, Y is instantiated as management
 - X is instantiated as biology, Y is instantiated as medicin

4.5 ? student(adrian).

Query type: ground.

Output: true.

Steps:

- **Unify** the query with the head of the rule:

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

- **Instantiate** adrian to X.

- **Resolve** into 1 goal:

```
student(adrian, _).
```

- The goal will be unified with

```
student(adrian, web_design).
```

- The goal proves true.

4.6 ? student(anna, engineering).

Query type: ground.

Output: true.

Steps:

- **Unify** the query with the head of the rule:

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

- **Instantiate** anna as X, engineering as Y.

- **Resolve** to 2 new queries:

```
field(Z, engineering), student(anna, Z).
```

- Both goals are evaluated.
- Both goals prove true with unification of:

```
field(hit_transfer, engineering), student(anna, hit_transfer).
```

4.7 ? student(X, engineering).

Query type: non-ground.

Output:

```
X = anna ;  
X = daniel ;  
X = adrian ;  
false.
```

Steps:

- **Unify** the query with the head of the rule:

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

- **Instantiate** engineering to Y.

- **Resolve** to 2 goals:

```
field(Z, engineering), student(X, Z).
```

- The 2 goals are unified with:

```
field(mechanical,engineering), student(X, mechanical).  
field(computer, engineering), student(X, computer).
```

- In each of the goal, there are further unifications and instantiations. The final is as belowed:

```
field(mechanical,engineering), student(anna, mechanical).  
field(mechanical,engineering), student(daniel, mechanical).  
field(computer,engineering), student(adrian, computer).
```

4.8 ? student(X,fine-arts), course(fine_arts, Y).

Query type: non-ground.

Output: false. **Steps:**

- **Unify** the first query with the head of rule:

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

- **Instantiate** fine-arts to Y.

- **Resolve** into 2 goals:

```
field(Z, fine-arts), student(X, Z)
```

- **Unify** the first goal with the head of rule:

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

- **Instantiate** Y with fine-arts.
- **Resolve** into 2 goals:

```
course(X, Z), field(Z, fine-arts).
```

- ...
- This is actually a recursion that will never end. Eventually, Prolog will result in *false* after exhausting the knowledge base.

4.9 ? field(_,X).

Query type: non-ground.

Output:

```
X = engineering ;  
X = engineering ;  
X = art ;  
X = social ;  
X = buisiness ;  
X = engineering ;  
X = engineering ;  
X = art ;  
X = social ;  
X = buisiness ;  
false.
```

Steps:

- **Unify** with facts respectively:

```
field(mechanical, engineering).  
field(computer, engineering).  
field(fine-arts, art).  
field(literature, social).  
field(management, buisiness).
```

- **Instantiate** X to engineering.
- **Instantiate** X to engineering.
- **Instantiate** X to art.
- **Instantiate** X to social.
- **Instantiate** X to buisiness.
- **Resolve**
- **Unify** with the head of rule:

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

- **Instantiate** X to all listed courses (hit_transfer, web_design, design_methods, poetry, leadership, biology respectively).

- **Resolve** into 2 goals:

```
course(X, Z), field(Z, Y).
```

- Evaluate both goals. The possible results are:

```
course(hit_transfer, mechanical), field(mechanical,
    engineering).
course(web_design, computer), field(computer, engineering).
course(design_methods, fine-arts), field(fine-arts, art).
course(poetry, literature), field(literature, social).
course(leadership, management), field(management, business).
```

- Output of this part is as followed:

```
X = engineering;
X = engineering;
X = art;
X = social;
X = business;
```

4.10 ? lab_number(____,X), field(X,Y).

Query type: non-ground.

Output: false.

Steps:

- **Unify** with the facts respectively:

```
lab_number(mechanical,15).
lab_number(fine_arts,10).
```

- This will **instantiate** X with a number, either 10 or 15.
- However, both will prove false in the second query when field(X, Y) is instantiated and resolved to either field(15, Y) or field(10, Y).
- Thus, Prolog outputs false.

4.11 ? lab_number(X,15), field(X,Y).

Query type: non-ground.

Output:

```
X = mechanical,  
Y = engineering ;  
X = hit_transfer,  
Y = engineering ;  
false.
```

Steps:

- **Unify** the first goal with the fact:

```
lab_number(mechanical,15).
```

- **Instantiate** mechanical to X.
- Unify the second goal with the fact:

```
field(mechanical,engineering).
```

- Get output: X = mechanical; Y = engineering

Steps when unify with rule:

- **Unify** with the head of rules:

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

- **Instantiate** 15 to Z.
- **Resolve** into 2 goals:

```
course(X, Y), lab_number(Y, 15).  
course(X, Z), field(Z, Y).
```

- Evaluate both goals results in unification as followed:

```
course(hit_transfer, mechanical), lab_number(mechanical, 15),  
field(mechanical, engineering).
```

- Thus, the output for this evaluation is: X = hit_transfer, Y = engineering.

4.12 ? student(X), !, student(X,_).

Query type: non-ground.

Output:

```
X = anna ;
X = anna ;
false.
```

Steps:

- **Unify** with the head of rule:

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

- **Resolve** the goal:

```
student(X, _).
```

- Evaluate the goal and found the first result of: $X = \text{Anna}$.
- Encounter **!**, stop evaluating.
- Continue to the third goal, **Instantiate** X with Anna from the first goal.
- **Unify** with the head of rule:

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

- **Instantiate** X with Anna.
- **Resolve** into 2 goals:

```
field(Z, Y), student(Anna, Z).
```

- Evaluate the goals, results in $X = \text{Anna}$ proves true.

4.13 ? student(X), student(X,_), !.

Query type: non-ground.

Output:

```
X = anna.
```

- **Unify** with the rule:

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

- **Resolve** into 1 goal:

```
student(X, _).
```

- Evaluate and unify with student(anna, hit_transfer)
- **Instantiate** X with anna. Produce output: $X = \text{anna}$.
- Reached the cut operator, stop.

4.14 ? `course(X,_)`, \+ `student(_,X)`.

Query type: non-ground.

Output:

`X = biology.`

Steps:

- **Unify** the first query firstly with the fact:

`course(hit_transfer, mechanical).`

- **Instantiate** X as `hit_transfer`,
- **Resolve** the query to be true.
- The second query will be unified with the fact:

`student(anna, hit_transfer).`

- the goals prove false. Backtracking..
- **Unify** the first query firstly with the fact:

`course(web_design, computer).`

- **Instantiate** X as `web_design`,
- **Resolve** the query to be true.
- The second query will be unified with the fact:

`student(adrian, web_design).`

- the goals prove false. Backtracking..
- **Unify** the first query firstly with the fact:

`course(design_methods, fine-arts).`

- **Instantiate** X as `design_methods`.
- **Resolve** the query to be true.
- The second query will be unified with the fact:

`student(ava, design_methods).`

- the goals prove false. Backtracking..

- **Unify** the first query firstly with the fact:

```
course(poetry, literature).
```

- **Instantiate** X as poetry.
- **Resolve** the query to be true.
- The second query will be unified with the fact:

```
student(jack, poetry).
```

- the goals prove false. Backtracking..
- **Unify** the first query firstly with the fact:

```
course(leadership, management).
```

- **Instantiate** X as leadership.
- **Resolve** the query to be true.
- The second query will be unified with the fact:

```
student(lee, leadership).
```

- the goals prove false. Backtracking..
- **Unify** the first query firstly with the fact:

```
course(biology, medicin).
```

- **Instantiate** X as biology.
- **Resolve** the query to be true.
- The second query could not be unified with the any fact, produce false.
- Not false = true.
- the goals prove true. Hence, output: X = biology.

5 Question 8

5.1 ? 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 ground 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 = dooby;  
Hermione = dooby;  
Hermione = hermione;  
Hermione = mcGonagall;  
Hermione = rita_skeeter;
```

5.2 ? 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.

6 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.