

Assignment 2

Question 1. Given the following sentences:

(a) Translate these sentences into predicate logic:

- (i) Marcus was a man: $\text{Man}(\text{Marcus})$
- (ii) Marcus was a Pompeian: $\text{Pompeian}(\text{Marcus})$
- (iii) All Pompeians are Romans: $\forall x (\text{Pompeian}(x) \rightarrow \text{Roman}(x))$
- (iv) Caesar was a ruler: $\text{Ruler}(\text{Caesar})$
- (v) All Romans were either loyal to Caesar or hated him:

$\forall x (\text{Roman}(x) \rightarrow \text{LoyalTo}(x, \text{Caesar}) \vee \text{Hates}(x, \text{Caesar}))$

- (vi) Everyone is loyal to someone: $\forall x \exists y \text{LoyalTo}(x, y)$
- (vii) People only try to assassinate rulers they are not loyal to:

$\forall x, y (\text{Person}(x) \wedge \text{Ruler}(y) \wedge \sim \text{LoyalTo}(x, y) \rightarrow \text{TriesToAssassinate}(x, y))$

- (viii) Marcus tried to assassinate Caesar: $\text{TriesToAssassinate}(\text{Marcus}, \text{Caesar})$
- (ix) All men are people: $\forall x (\text{Man}(x) \rightarrow \text{Person}(x))$

(b) Convert the above sentence represented in predicate logic into Horn Clauses

- (i) Marcus was a man: $\text{Man}(\text{Marcus})$
- (ii) Marcus was a Pompeian: $\text{Pompeian}(\text{Marcus})$

(iii) All Pompeians are Romans: $\forall x (\text{Roman}(x) \leftarrow \text{Pompeian}(x))$

(iv) Caesar was a ruler: $\text{Ruler}(\text{Caesar})$

(v) All Romans were either loyal to Caesar or hated him:

$\text{LoyalTo}(x, \text{Caesar}) \vee \text{Hates}(x, \text{Caesar}) \leftarrow \text{Roman}(x)$

(vi) Everyone is loyal to someone: $\text{LoyalTo}(x, y) \leftarrow \text{Person}(x)$

(vii) People only try to assassinate rulers they are not loyal to:

$\text{TriesToAssassinate}(x, y) \leftarrow \text{Person}(x), \text{Ruler}(y), \sim \text{LoyalTo}(x, y)$

(viii) Marcus tried to assassinate Caesar: $\text{TriesToAssassinate}(\text{Marcus}, \text{Caesar})$

(ix) All men are people: $\text{Person}(x) \leftarrow \text{Man}(x)$

(c) Write the horn clauses into Prolog syntax on any Prolog system

```
HelloWorld.pl
1 :- initialization(main).
2
3 man(marcus).
4 pompeian(marcus).
5 roman(X) :- pompeian(X).
6 ruler(caesar).
7 person(X) :- man(X).
8
9 % Everyone is loyal to someone (assumption)
10 loyalto(X, _) :- person(X).
11
12 % People only try to assassinate rulers they are not loyal to
13 tried_to_assassinate(X, Y) :- person(X), ruler(Y), \+ loyalto(X, Y).
14
15 % Marcus tried to assassinate Caesar
16 tried_to_assassinate(marcus, caesar).
17
18 % All Romans are either loyal to Caesar or hate him
19 hates(X, caesar) :- roman(X), \+ loyalto(X, caesar).
20
21 main :-
22     write('Knowledge base loaded successfully!'), nl.
23
```

Output:

Knowledge base loaded successfully!

(d) Prove that Marcus hates Caesar by submitting this predicate as a goal to your KB

```
HelloWorld.pl 43b5k4num
1 :- initialization(main).
2
3 % Facts
4 man(marcus).
5 pompeian(marcus).
6 ruler(caesar).
7
8 % Rules
9 roman(X) :- pompeian(X).
10 person(X) :- man(X).
11
12 % Explicit loyalty facts (Marcus is NOT in this list)
13 loyalto(john, caesar). % Example: John is loyal to Caesar
14
15 % People only try to assassinate rulers they are not loyal to
16 tried_to_assassinate(X, Y) :- person(X), ruler(Y), \+ loyalto(X, Y).
17
18 % Marcus tried to assassinate Caesar
19 tried_to_assassinate(marcus, caesar).
20
21 % All Romans either are loyal to Caesar or hate him
22 hates(X, caesar) :- roman(X), \+ loyalto(X, caesar).
23
24 % Main function to confirm the program loads successfully
25 main :-
26     write('Knowledge base loaded successfully!'), nl,
27     (hates(marcus, caesar) -> write('Marcus hates Caesar: true'), nl ; write('Marcus hates Caesar: false'), nl).
28
```

Output:

Knowledge base loaded successfully!

Marcus hates Caesar: true

Question 2. Given the following sentences:

(a) Translate these sentences into predicate logic:

- (i) Jack owns Fiat: Owns(Jack, Fiat)
- (ii) John owns Opel: Owns(John, Opel)
- (iii) Fiat is a car and Opel is a car too: Car(Fiat), Car(Opel)
- (iv) Every car owner can drive: $\forall x(\text{Owns}(x, y) \wedge \text{Car}(y) \rightarrow \text{CanDrive}(x))$
- (v) Jack exceeds the speed limit: ExceedsSpeedLimit(Jack)
- (vi) John fastens the seat belt: FastenSeatBelt(John)

- (vii) Every driver who exceeds the speed limit or does not fasten a seat belt breaks traffic rules:

$$\forall x (\text{CanDrive}(x) \wedge (\text{ExceedsSpeedLimit}(x) \vee \sim \text{FastenSeatBelt}(x)) \rightarrow \text{BreaksTrafficRules}(x))$$

- (viii) Everyone who can drive is a driver: $\forall x (\text{CanDrive}(x) \rightarrow \text{Driver}(x))$

- (ix) Everyone who breaks any traffic rules will get a fine:

$$\forall x (\text{BreaksTrafficRules}(x) \rightarrow \text{GetsFine}(x))$$

- (x) Bad drivers get fines: $\forall x (\text{BadDriver}(x) \rightarrow \text{GetsFine}(x))$

- (xi) Good drivers do not get fines: $\forall x (\text{GoodDriver}(x) \rightarrow \sim \text{GetsFine}(x))$

(b) Convert the above sentence represented in predicate logic into Horn Clauses

- (i) Jack owns Fiat: $\text{Owns}(\text{Jack}, \text{Fiat})$
- (ii) John owns Opel: $\text{Owns}(\text{John}, \text{Opel})$
- (iii) Fiat is a car and Opel is a car too: $\text{Car}(\text{Fiat}), \text{Car}(\text{Opel})$
- (iv) Every car owner can drive: $\text{CanDrive}(x) \leftarrow \text{Owns}(x, y), \text{Car}(y)$
- (v) Jack exceeds the speed limit: $\text{ExceedsSpeedLimit}(\text{Jack})$
- (vi) John fastens the seat belt: $\text{FastenSeatBelt}(\text{John})$
- (vii) Every driver who exceeds the speed limit or does not fasten a seat belt breaks traffic rules:

$$\text{BreaksTrafficRules}(x) \leftarrow \text{CanDrive}(x), \text{ExceedsSpeedLimit}(x)$$

$$\text{BreaksTrafficRules}(x) \leftarrow \text{CanDrive}(x), \sim \text{FastenSeatBelt}(x)$$


- (viii) Everyone who can drive is a driver: $\text{Driver}(x) \leftarrow \text{CanDrive}(x)$

- (ix) Everyone who breaks any traffic rules will get a fine:

$$\text{GetsFine}(x) \leftarrow \text{BreaksTrafficRules}(x)$$

- (x) Bad drivers get fines: $\text{GetsFine}(x) \leftarrow \text{BadDriver}(x)$
- (xi) Good drivers do not get fines: $\sim\text{GetsFine}(x) \leftarrow \text{GoodDriver}(x)$

(c) Write the horn clauses into Prolog syntax on any Prolog system available online

```
HelloWorld.pl 43b5k4num 
1 :- initialization(main).
2
3 % Facts: Ownership of cars
4 owns(jack, fiat).
5 owns(john, opel).
6
7 % Facts: Cars
8 car(fiat).
9 car(opel).
10
11 % Rule: If someone owns a car, they can drive
12 can_drive(X) :- owns(X, Y), car(Y).
13
14 % Facts: Driving behavior
15 exceeds_speed_limit(jack).
16 fasten_seat_belt(john).
17
18 % Rule: Every car owner is a driver
19 driver(X) :- can_drive(X).
20
21 % Rule: Drivers break traffic rules if they exceed speed limit OR don't fasten seat belt
22 breaks_traffic_rules(X) :- can_drive(X), exceeds_speed_limit(X).
23 breaks_traffic_rules(X) :- can_drive(X), \+ fasten_seat_belt(X).
24
25 % Rule: Anyone who breaks traffic rules gets a fine
26 gets_fine(X) :- breaks_traffic_rules(X).
27
28 % Rule: Bad drivers get fines
29 bad_driver(X) :- gets_fine(X).
30
31 % Rule: Good drivers do not get fines
32 good_driver(X) :- \+ gets_fine(X).
33
34 % Main function just to confirm the knowledge base loads
35 main :- write('Knowledge base loaded successfully!'), nl.
36
```

Output:

Knowledge base loaded successfully!

(d) Prove that Jack is a bad driver and John is a good driver by submitting each of the two previous predicates as goals to your knowledge base. If the KB failed to infer any of these goals, explain why it failed and what predicate should be there to prove your goals.

```
HelloWorld.pl 43b5k4num
1 :- initialization(main).
2
3 % Reuse the same knowledge base from 2(c)
4 owns(jack, fiat).
5 owns(john, opel).
6
7 car(fiat).
8 car(opel).
9
10 can_drive(X) :- owns(X, Y), car(Y).
11
12 exceeds_speed_limit(jack).
13 fasten_seat_belt(john).
14
15 driver(X) :- can_drive(X).
16
17 breaks_traffic_rules(X) :- can_drive(X), exceeds_speed_limit(X).
18 breaks_traffic_rules(X) :- can_drive(X), \+ fasten_seat_belt(X).
19
20 gets_fine(X) :- breaks_traffic_rules(X).
21
22 bad_driver(X) :- gets_fine(X).
23 good_driver(X) :- \+ gets_fine(X).
24
25 % Main function to prove Jack is a bad driver & John is a good driver
26 main :-
27     write('Knowledge base loaded successfully!'), nl,
28     (bad_driver(jack) -> write('Jack is a bad driver: true'), nl ; write('Jack is a bad driver: false'), nl),
29     (good_driver(john) -> write('John is a good driver: true'), nl ; write('John is a good driver: false'), nl).
30
31
```

Output:

Knowledge base loaded successfully!

Jack is a bad driver: true

John is a good driver: true