Exercise 1 (Representation in Prolog - 19 pts).

1.1)

```
science_fiction_novel(i_robot).
```

1.2)

```
film(i_robot).
```

1.3)

```
written_by(X, isaac_asimov) :- science_fiction_novel(X).
science_fiction_writer(isaac_asimov).
```

1.4)

```
human(spooner).
robot(sonny).
injure_human(spooner).
harm_human(sonny, spooner).

firstLaw(Human, Robot) :-
    human(Human),
    robot(Robot),
    not(injure_human(Human)),
    not(harm_human(Robot,Human)).
```

1.5)

```
obey(sonny, spooner).

secondLaw(Human, Robot) :-
    human(Human),
    robot(Robot),
    not(obey(Robot, Human)),
    not(firstLaw(Human, Robot)).
```

1.6)

```
protect_self(sonny).

thirdLaw(Human, Robot) :-
    human(Human),
    robot(Robot),
    not(protect_self(Robot)),
    not(firstLaw(Human, Robot)),
    not(secondLaw(Human, Robot)).
```

1.7)

```
human(spooner).
human(will).
human(smith).

robot(sonny).
robot(danger).
robot(humanoid).

injure_human(spooner).

harm_human(sonny, spooner).
harm_human(will, smith).

obey(sonny, spooner).
obey(danger, will).
obey(humanoid, smith).

protect_self(sonny).
protect_self(danger).
protect_self(humanoid).
```

Exercise 2 (Unification - 16.5 pts).

2.1) X = f(Y).

2.2) false.

2.3) Apple = Pear.

2.4) Pear = 'Apple'.

2.5) X = eenie,
Y = meenie.

2.6) false.

2.7) false.

2.8) X = ingredient(flower),
Y = topping(syrup).

2.9) false.

2.10) X = pancake(X).

2.11) X = pancake(X).

Exercise 3 (ordering -11.5 pts).

3.1)

```
summer(X):-
    happy(X).

warm(Y) :-
    warm(Y).

warm(a).

summer(X):-
    warm(Y).

happy(b).
```

Line 1-2 will search for a match with happy in the last line. Line 5-6 will search for warm, which will be found in line 5-6 creating a non-stopping loop.

3.2)

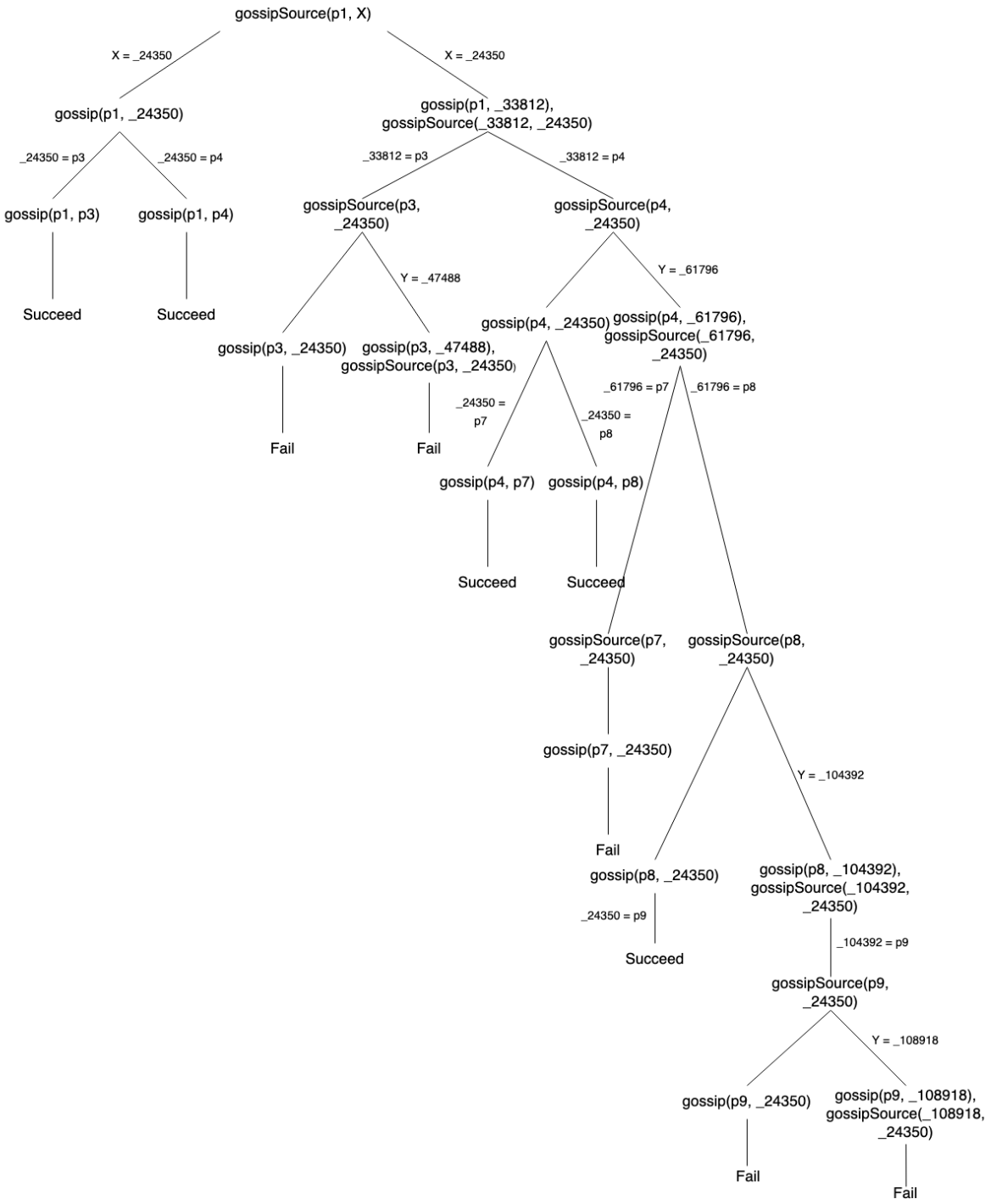
```
warm(a).
happy(b).

warm(Y) :-
    warm(Y).

summer(X):-
    happy(X).

summer(X):-
       warm(Y).
```

Line 1-2, variables are defined. Line 4-5 searches for warm, line 7-8 will be set up with line 2 and line 10-11 will find line 1, thus because both variables have been found resulting in an output of repeating trues.

Exercise 4 (Prolog search -20 pts).

4.1) Rule:
gossipSource(X, Y) :- gossip(X, Y) ; (gossip(X, Z) , gossipSource(Z, Y)).

4.2) The search tree is as follows:



4.3) Fact:
gossip(X, Y).

Exercise 5 (negation -16 pts).

5.1) Rule:
enjoys(vincent ,X) :- not(bigKahunaBurger(X)), burger(X).

Query:
?- enjoys(vincent ,X).

Explanation of outcome:
In prolog, the compiler first matches the rule to the Knowledge base. Once that is done, it starts comparing the body of the rule to the information given in the knowledge base. The order of this comparison is from left to right. In our case, once “enjoys” is compared, the next comparison being made is “bigKahunaBurger”. According to the data, bigKahunaBurger(b). in the knowledge base, X gets the value b. Now, the not() function gives us the opposite of bigKahunaBurger(b), that is False. Hence, as the rule is in conjunction, the outcome will always be False.

5.2) Changed rule:
enjoys(vincent, X) :- burger(X), not(bigKahunaBurger(X)).

5.3) Atoms returned:
X = a ;
X = c ;
X = d.

Explanation of outcome:
As explained in 5.1, the order of comparing the rule’s body is from left to right. Now, as burger(X) appears before not(bigKahunaBurger(X)), the compiler first compares the burger. So, X can have the values a, c, b and d. However, once the compiler compares bigKahunaBurger and checks the not() function, the outcome of X = b will become false. Therefore, X = b is not returned as an atom.

Exercise 6 (recursion -21 pts).

6.1) Rule and clauses:
subtract(X,0,X).
subtract(succ(X), succ(Y), Z) :- subtract(X,Y,Z).

6.2) Rule and clauses:
subtract(X,0,X).
subtract(0, X, -X).
subtract(succ(X), succ(Y), Z) :- subtract(X,Y,Z).

6.3) ?- subtract(0,Y,Z).
The compiler compares this query to subtract(X, 0, X). in the knowledge base. So, X = 0, 0 = Y, and X = Z. As X = 0, Z = 0 as well. Then, the compiler tries to unify the query with the rule subtract(succ(X), succ(Y), Z) :- subtract(X,Y,Z). As succ(X) = 0 is not possible, the output does not show fail and the end is reached.

?- subtract(X,0,Z) **.**
The compiler compares this query to subtract(X, 0, X). in the knowledge base. So, X = X, 0 = 0, and X = Z. Next, the query is compared to subtract(0, X, -X). Now, 0 = X , X = 0, -X = Z. As X = 0, Z = 0 as well. Then, the compiler tries to unify the query with the rule subtract(succ(X), succ(Y), Z) :- subtract(X,Y,Z). Instantiating X to succ(X) is possible, but instantiating 0 to succ(Y) is not. So, the compilation fails and outputs false.

?- subtract(X,Y,0).

The compiler compares this query to subtract(X, 0, X). in the knowledge base. So, X = X, 0 = Y, and X = 0. Next, the query is compared to subtract(0, X, -X). Now, 0 = X , X = Y, -X = 0. As -X = 0, Y = 0 as well. Then, the compiler tries to unify the query with the rule subtract(succ(X), succ(Y), Z) :- subtract(X,Y,Z). Instantiating succ(X) to X is possible, instantiating Y to succ(Y) is possible too. Finally, 0 is instantiated to Z which means Z = 0 which in turn implies X = Y. Therefore, X and Y keep getting incremented by 1 but return True always as X = Y.