Expert System Practical Part:

Diagnose Car Issues:

Let's define some predicates:

Symptom(x, y): Predicate indicating that symptom y is associated with category x.
Diagnosis(x, z): Predicate indicating that the diagnosis for category x is z.

Now, we can represent the rules:

1.Engine category rules:

Symptom(Engine, "Engine won't start").
Symptom(Engine, "Strange noises").
Diagnosis(Engine, "Check the battery, starter, and alternator.").

2.Transmission category rules:

Symptom(Transmission, "Slipping gears").
Symptom(Transmission, "Burning smell").
Diagnosis(Transmission, "Inspect the transmission fluid and check for leaks.").

3.Brakes category rules:

Symptom(Brakes, "Squeaking noise").
Symptom(Brakes, "Soft brake pedal").
Diagnosis(Brakes, "Check brake pads, rotors, and brake fluid.")

Facts Section:

Issue(Engine)
Issue(Transmission)
Issue(Brakes)

Symptom(Engine, "Engine won’t start")
Symptom(Engine, "Strange noises")
Diagnosis(Engine, "Check the battery, starter, and alternator.")

Symptom(Transmission, "Slipping gears")
Symptom(Transmission, "Burning smell")
Diagnosis(Transmission, "Inspect the transmission fluid and check for leaks.")

Symptom(Brakes, "Squeaking noise")
Symptom(Brakes, "Soft brake pedal")
Diagnosis(Brakes, "Check brake pads, rotors, and brake fluid.")

Python Code is shown below:

```python
class ExpertSystem:
    def __init__(self):
        self.knowledge_base = {
            'engine': {
                'symptoms': ['engine not start', 'strange noises'],
                'diagnosis': 'Check the battery, starter, and alternator.'
            },
            'transmission': {
                'symptoms': ['slipping gears', 'burning smell'],
                'diagnosis': 'Inspect the transmission fluid and check for leaks.'
            },
            'brakes': {
                'symptoms': ['squeaking noise', 'soft brake pedal'],
                'diagnosis': 'Check brake pads, rotors, and brake fluid.'
            }
            # Add more categories and rules as needed
        }

    def diagnose(self, issue, symptoms):
        issue = issue.lower()
        if issue in self.knowledge_base:
          if all(symptom.lower() in self.knowledge_base[issue]['symptoms'] for
                 symptom in symptoms):
                return self.knowledge_base[issue]['diagnosis']
        return "Unable to diagnose the issue."

def ask_user_for_issue():
    print("Select an issue:")
    for issue in expert_system.knowledge_base.keys():
        print(f"- {issue}")
    return input("Enter the chosen issue: ").lower()

def ask_user_for_symptoms():
    symptoms = []
    while True:
        symptom = input("Enter a symptom (or 'done' to finish): ").lower()
        if symptom == 'done':
            break
        symptoms.append(symptom)
    return symptoms

```

```
46      def main():
47          issue = ask_user_for_issue()
48          symptoms = ask_user_for_symptoms()
49
50          diagnosis = expert_system.diagnose(issue, symptoms)
51
52          print("\nDiagnosis:")
53          print(diagnosis)
54
55      if __name__ == "__main__":
56          expert_system = ExpertSystem()
57          main()
58
```

An expert system for diagnosing computer issues.

Facts Section:

Category(Hardware).
Category(Software).

Symptom(Hardware, "Computer doesn't power on").
Symptom(Hardware, "Unusual noises from the computer").

Symptom(Software, "Frequent program crashes").
Symptom(Software, "Slow computer performance").

```
Diagnosis(Hardware, "Check power supply and internal hardware components.")
Diagnosis(Software, "Scan for malware and optimize software performance.")
```

Rule Base Section:

```
DiagnoseIssue(Issue, Symptom) :-
                Symptom(Issue, Symptom),
                Diagnosis(Issue, _).

CategoryIssue(Category, Issue) :-
                 Category(Issue, _).

DiagnoseCategory(Category) :-
                    CategoryIssue(Category, Symptom),
                    write("Diagnosis for "), write(Category), write(": "),
                    DiagnoseIssue(Category, Symptom), nl.

DiagnoseAllCategories :-
   Category(Issue, _),
   DiagnoseCategory(Issue),
   !.

DiagnoseAllCategories.
```

Use quantifiers and FOL (Predicate Logic) to represent the following:

1. Lily loves to play in the park.
2. Lily is afraid of dog.
3. Dog barks and growls.
4. Lily runs away from dog.
5. Lily goes back to the park.
6. Lily walks slowly up to dog.
7. Dog stops barking and growling.
8. Lily reaches out her hand.
9. Dog sniffs Lily's hand.
10. Dog licks Lily's hand.
11. Lily is not afraid of dog.
12. Lily and dog become friends.
13. Lily plays with dog.
14. Dog is not scary.

Solution:

1. Lily loves to play in the park.

$\exists P\ [park(P) \wedge LovesPlayInPark(Lily, P)]$ الحل

Park(p): p is a park.

LovesPlayInPark(x, p): x loves to play in park p.

2. Lily is afraid of dog.

$\exists X [dog(X) \wedge afraidOf(lily, X)]$

dog(X): X is a dog.

afraidOf(Y, X): Y is afraid of X.

3. Dog barks and growls.

$\forall X [dog(X) \rightarrow barks(X) \wedge growls(X)]$

4. Lily runs away from dog.

$\exists X [dog(X) \wedge lilyRunsAwayFrom(lily, X)]$

5. Lily goes back to the park.

$\exists X [Parks(X) \wedge GoesToPark(lily, X) \wedge HasBeenToPark(lily,X)]$ حل اول

$\exists x [Park(x) \wedge GoesToPark(Lily, x)]$ حل ثاني

6. Lily walks slowly up to dog.

$\exists x [Dog(x) \wedge WalksSlowlyUpTo(Lily, x)]$ حل اول

$\exists Dog [dog(Dog) \wedge Lily(Lily) \wedge WalksSlowlyUpTo(Lily, Dog)]$ حل ثاني

7. Dog stops barking and growling.

$\exists x [Dog(x) \wedge (Barks(x) \wedge Growls(x)) \rightarrow StopsBarkingGrowling(x)]$ حل اول

$\exists x [Dog(x) \wedge BarksAndGrowls(x) \rightarrow StopsBarkingAndGrowling(x)]$ حل ثاني

$\exists x [Dog(x) \wedge StopsBarkingGrowling(x)]$ حل ثالث

8. Lily reaches out her hand.

$\exists x [Lily(x) \land ReachesOut(x)]$ حل اول

$\exists h [ReachesOut(Lily, h) \land Hand(h)]$ حل ثاني

9. Dog sniffs Lily's hand.

$\exists x [Dog(x) \land SniffsHand(x, Lily)]$

10. Dog licks Lily's hand.

$\exists x [Dog(x) \land LicksHand(x, Lily)]$

11. Lily is not afraid of dog.

$\neg AfraidOf(Lily, dog)$ حل اول

$\exists x [Dog(x) \land Not(AfraidOf(Lily, x))]$ حل ثاني

12. Lily and dog become friends.

$\exists x [Dog(x) \land Friends(Lily, x)]$

13. Lily plays with dog.

$\exists x [Dog(x) \land PlaysWith(Lily, x)]$

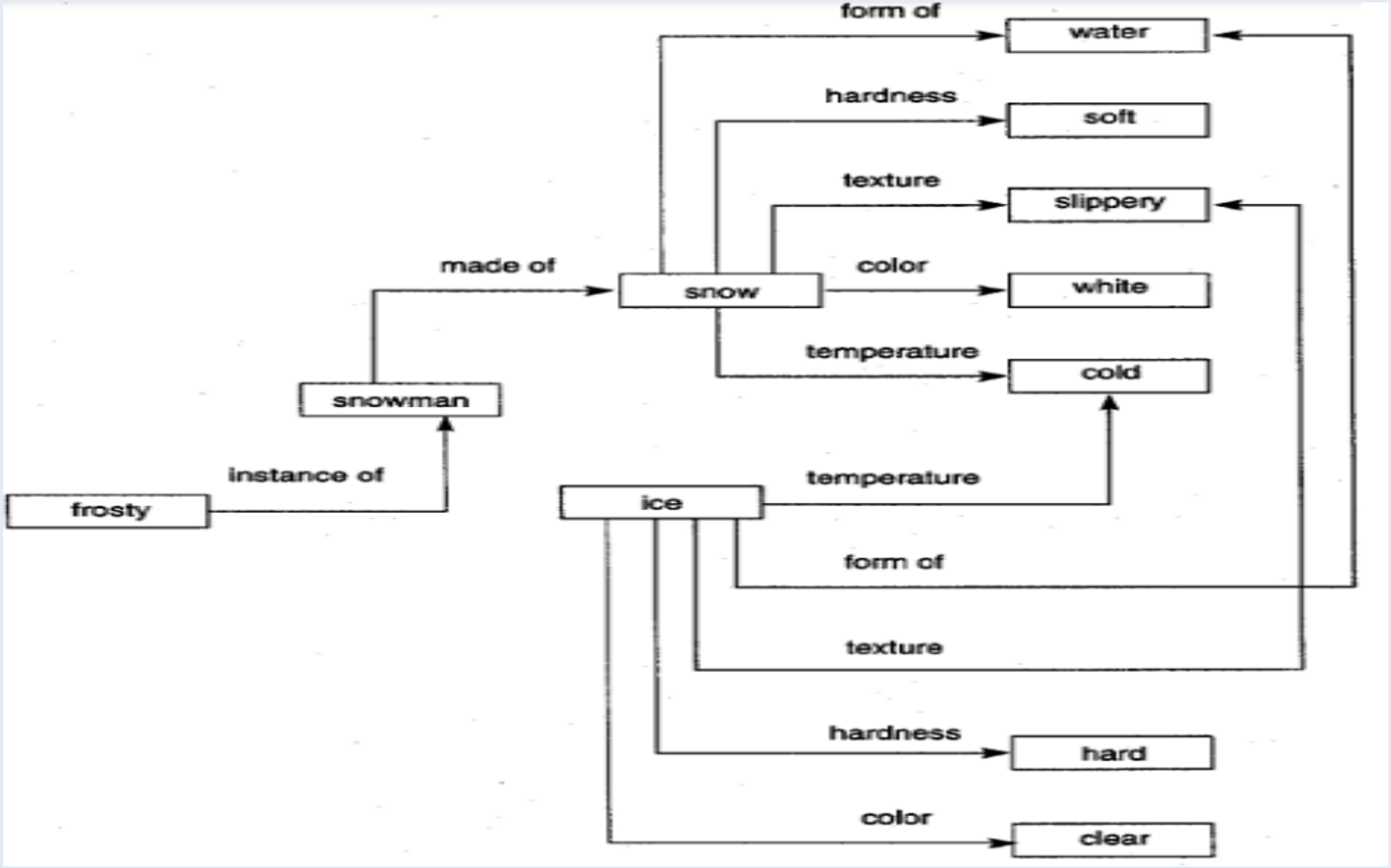
14. Dog is not scary.

$\forall x [Dog(x) \rightarrow NotScary(x)]$

Semantic Networks:

Example#1:

"Frosty is a snowman. A snowman is made of snow. Snow is frozen water. It is slippery and soft. Snow is cold. Ice is also cold. It is also frozen water, but unlike snow, which is soft, ice is hard. Ice is clear in color."



Example#2:

If you have the following knowledge. Use semantic networks for representation.

(1) A trout is a fish.

(2) A fish has gills.

(3) A fish has fins.

(4) Fish is food.

(5) Fish is animal.

(6) An apple is a fruit.

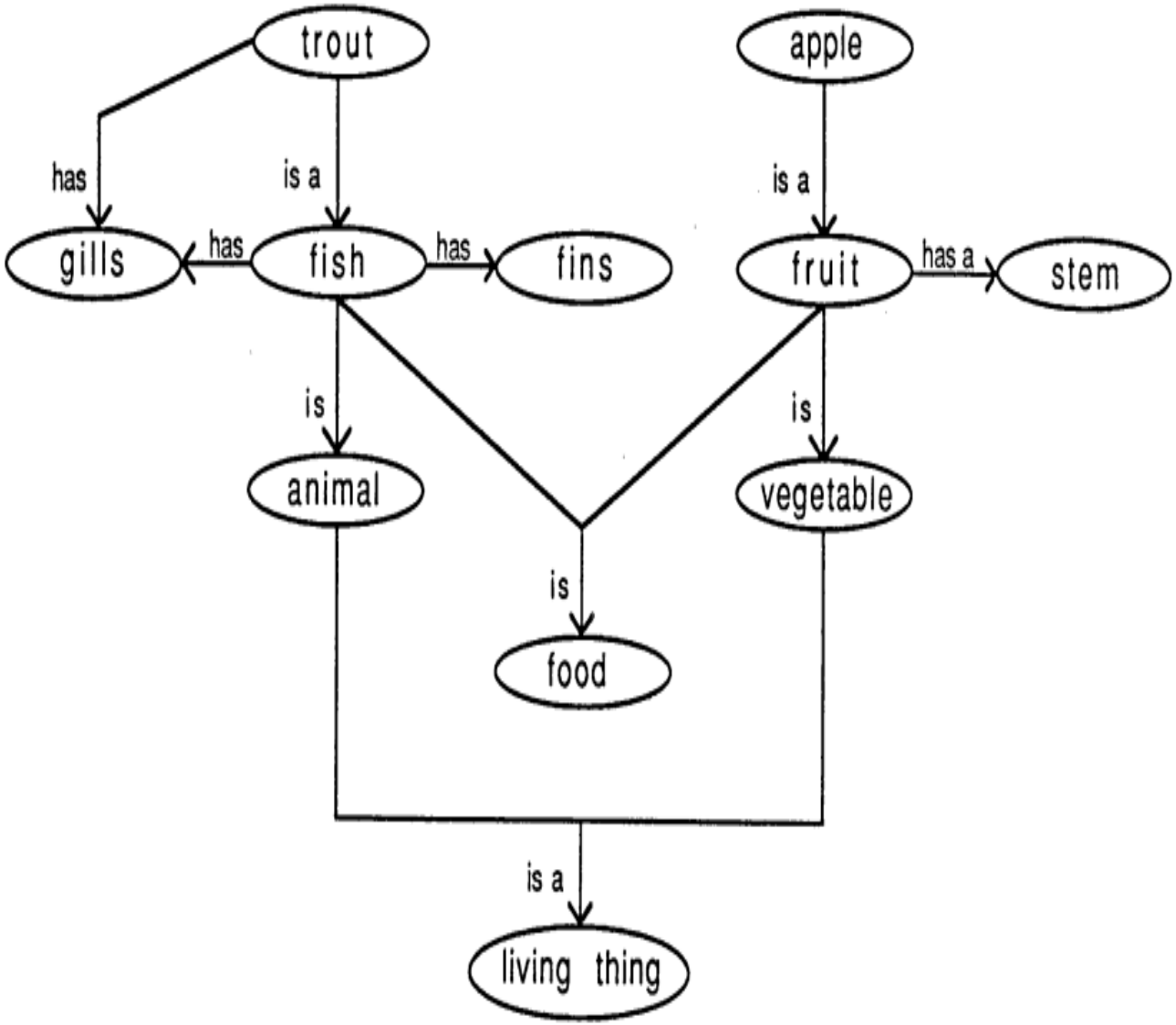
(7) Fruit has a stem.

(8) Fruit is food.

(9) Fruit is vegetable.

(10) An animal is a living thing.

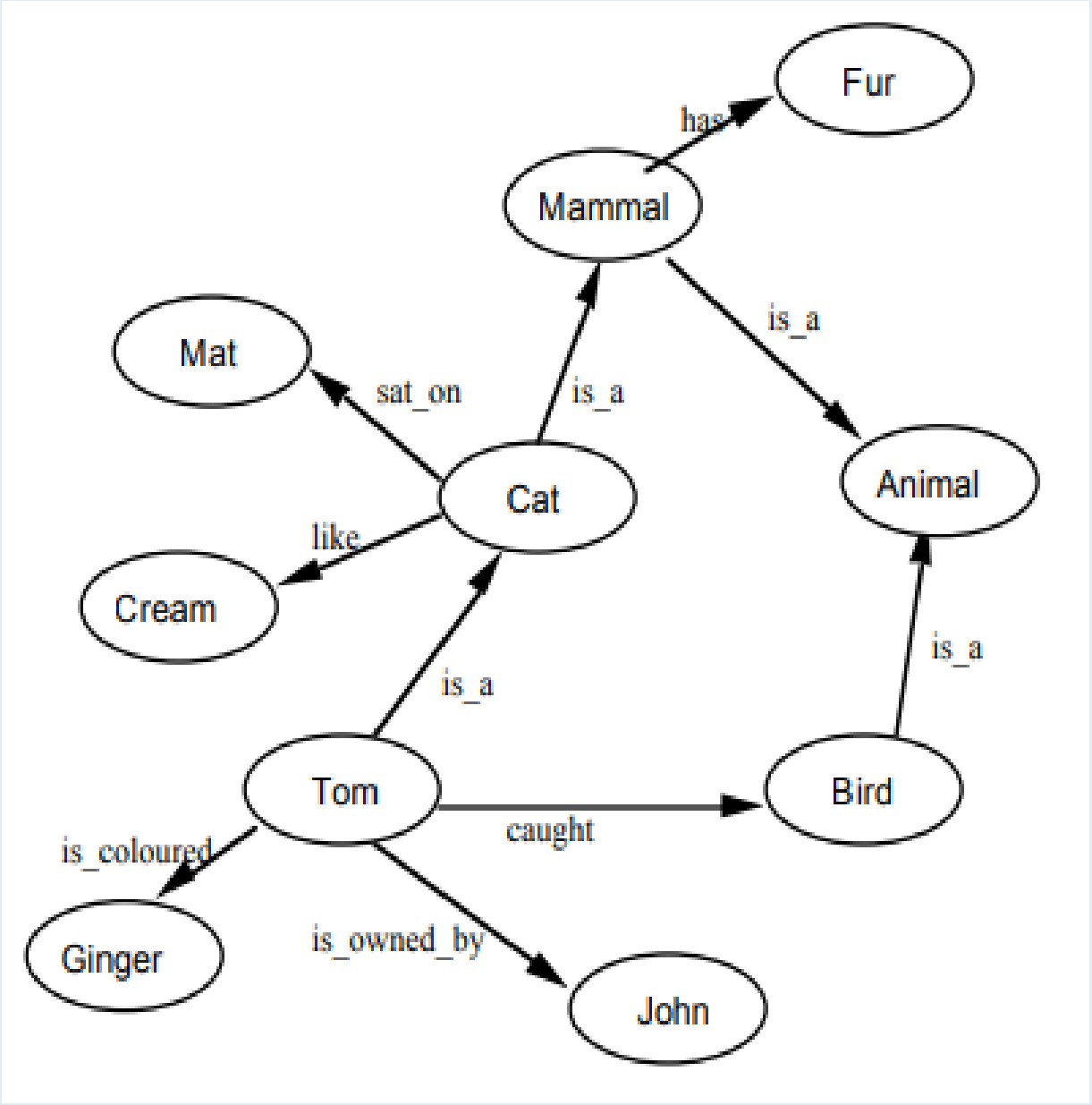
(11) A vegetable is a living thing.



Example#3:

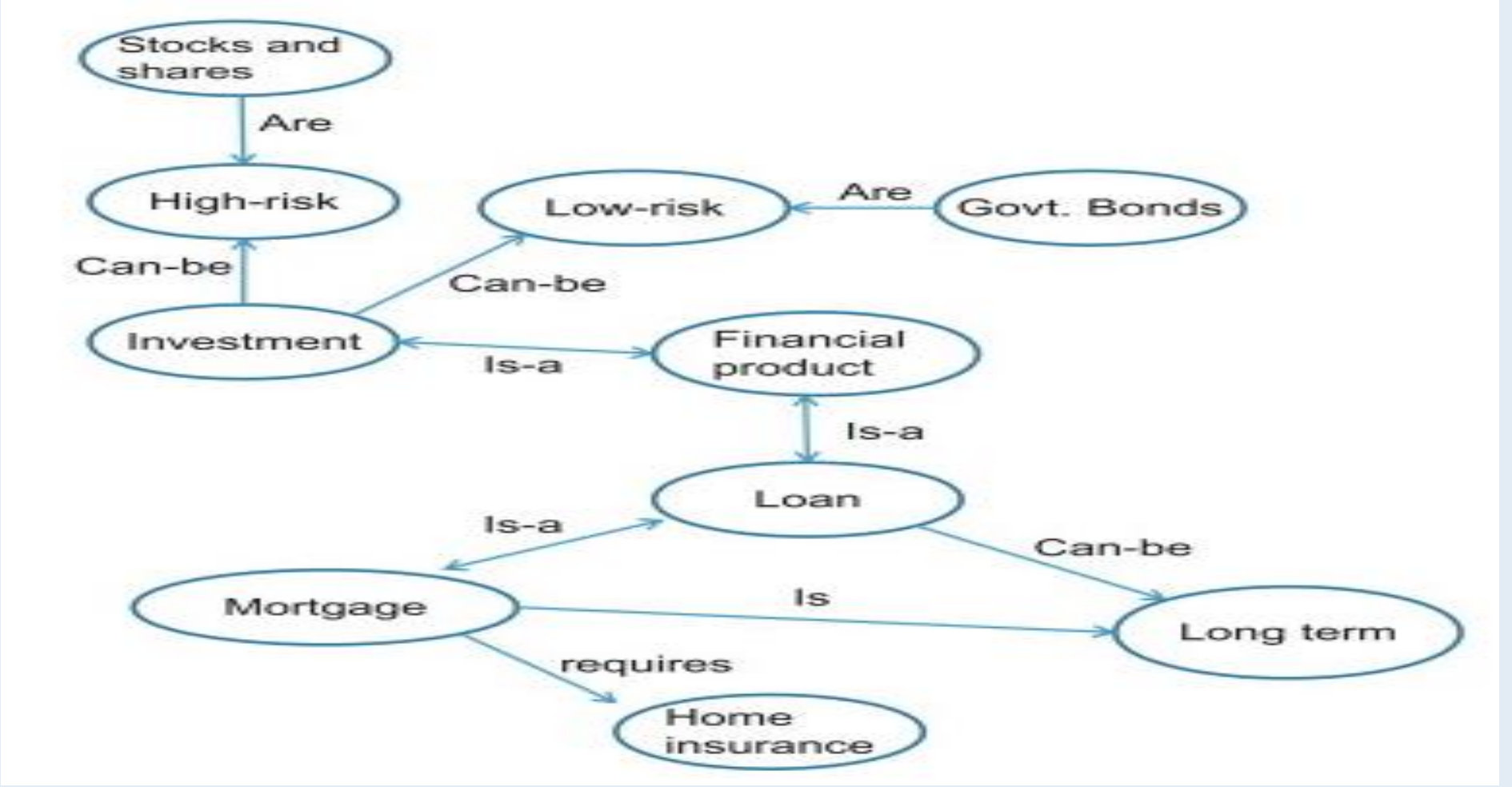
If you have the following knowledge. Use semantic networks for representation.

1- Tom is a cat.
2- Tom caught a bird.
3- Tom is owned by John.
4- Tom is ginger in colour.
5- Cats like cream.
6- The cat sat on the mat.
7- A cat is a mammal.
8- A bird is an animal.
9- All mammals are animals.
10- Mammals have fur.

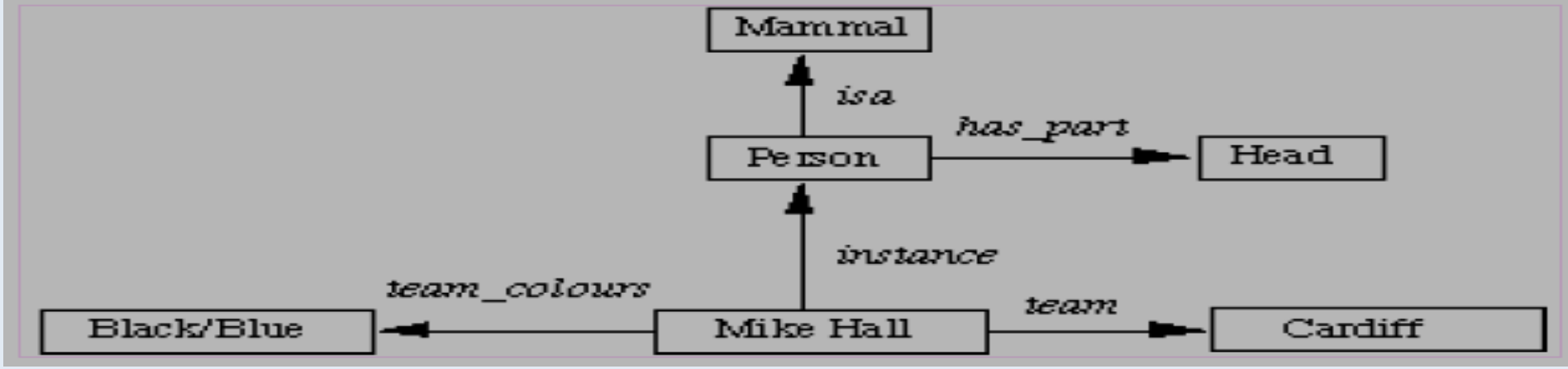


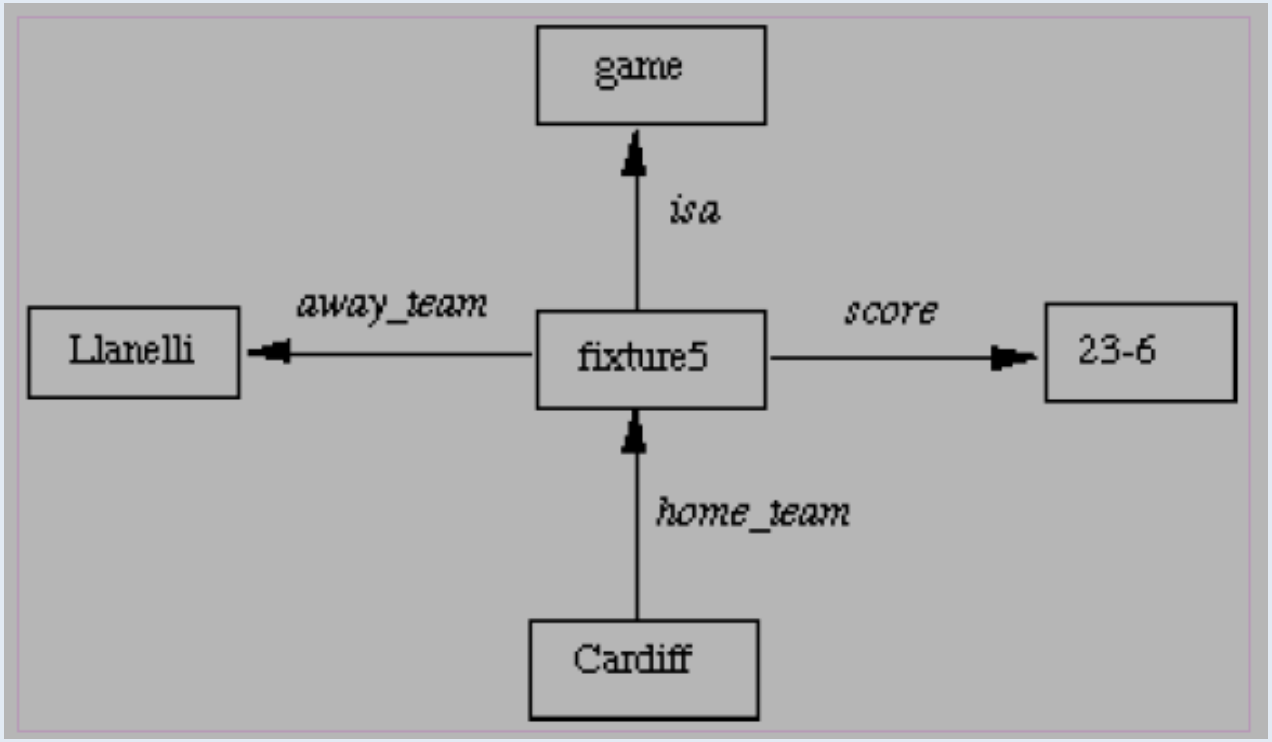
Example#4:

financial products can be described by their duration, risk level, and other characteristics: a mortgage is a financial product that is long-term and has a dependency on other financial products such as home insurance, life insurance, and current accounts. Figure 4.10 illustrates a simple semantic network where the principal concept is "financial product," which can be a loan or an investment. An example of a long-term loan is a mortgage, and an example of a low-risk investment is a government bond.

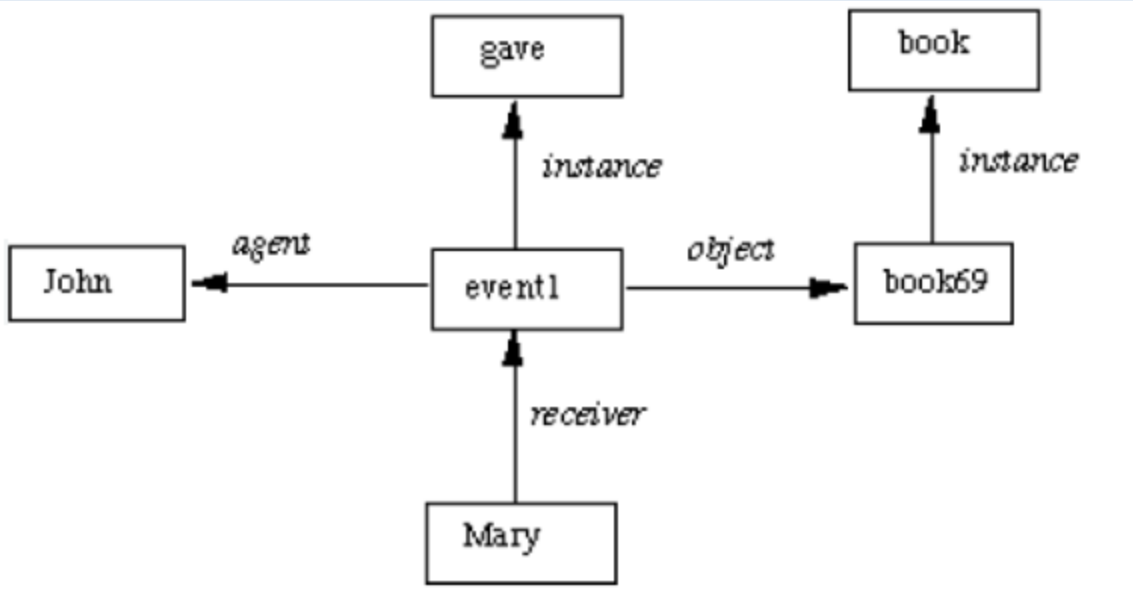


These values can also be represented in logic as: isa(person, mammal), instance(Mike-Hall, person) team(Mike-Hall, Cardiff)

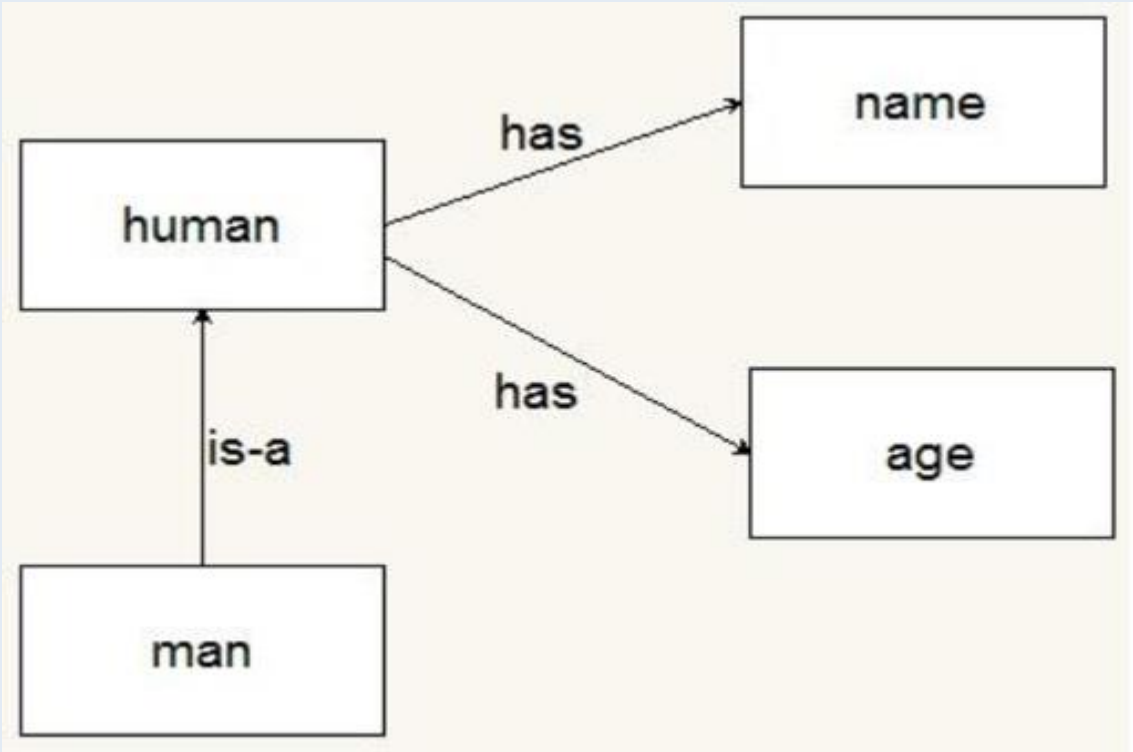




score(Cardiff, Llanelli, 23-6)



John gave Mary the book.



```python
def give(agent, recipient, object):
  """
  This function represents the predicate "give" in Python.

  Args:
    agent: The agent who gives (e.g., "John").
    recipient: The recipient of the object (e.g., "Mary").
    object: The object being given (e.g., "book").

  Returns:
    None

  """
  # You can add logic here to perform actions based on the arguments
  print(f"{agent} gives {object} to {recipient}.")

# Example usage
john_gives_book_to_mary = give("John", "Mary", "book")
```

Q5: Use semantic network to represent the following: “The dog Tom caught a cat. Cats considered being animals. All animals are mammals. Mammal has fur. Dogs sat on mat and they like bones. Tom’s color is brown and it is owned by Suha”.