Title: Rule based inference problems

EX. NO : 08 Name: Parth Langalia

DATE : 27-03-2023 **Reg No. :** RA2011033010033

AIM :

Developing an optimized technique using an appropriate artificial intelligence algorithm to detect the animal.

PSEUDO CODE :

In artificial intelligence, we need intelligent computers which can create new logic from old logic or by evidence, so generating the conclusions from evidence and facts is termed as Inference. Inference rules are the templates for generating valid arguments.

Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal.

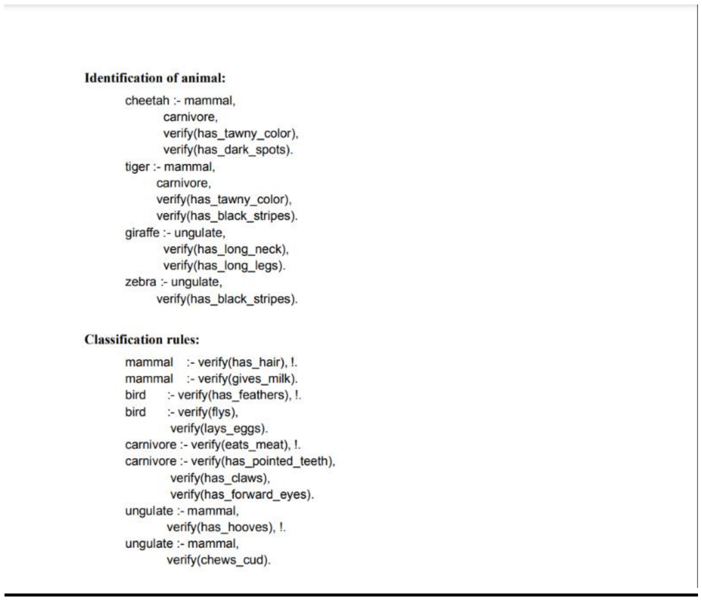
In inference rules, the implication among all the connectives plays an important role. Following are some terminologies related to inference rules:

● Implication: It is one of the logical connectives which can be represented as $P \rightarrow Q$. It is a Boolean expression.

● Converse: The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as $Q \rightarrow P$.

● Contrapositive: The negation of converse is termed as contrapositive, and it can be represented as $\neg Q \rightarrow \neg P$.

● Inverse: The negation of implication is called inverse.



Identification of animal:

```
cheetah :- mammal,
          carnivore,
          verify(has_tawny_color),
          verify(has_dark_spots).
tiger :- mammal,
        carnivore,
        verify(has_tawny_color),
        verify(has_black_stripes).
giraffe :- ungulate,
          verify(has_long_neck),
          verify(has_long_legs).
zebra :- ungulate,
        verify(has_black_stripes).
```

Classification rules:

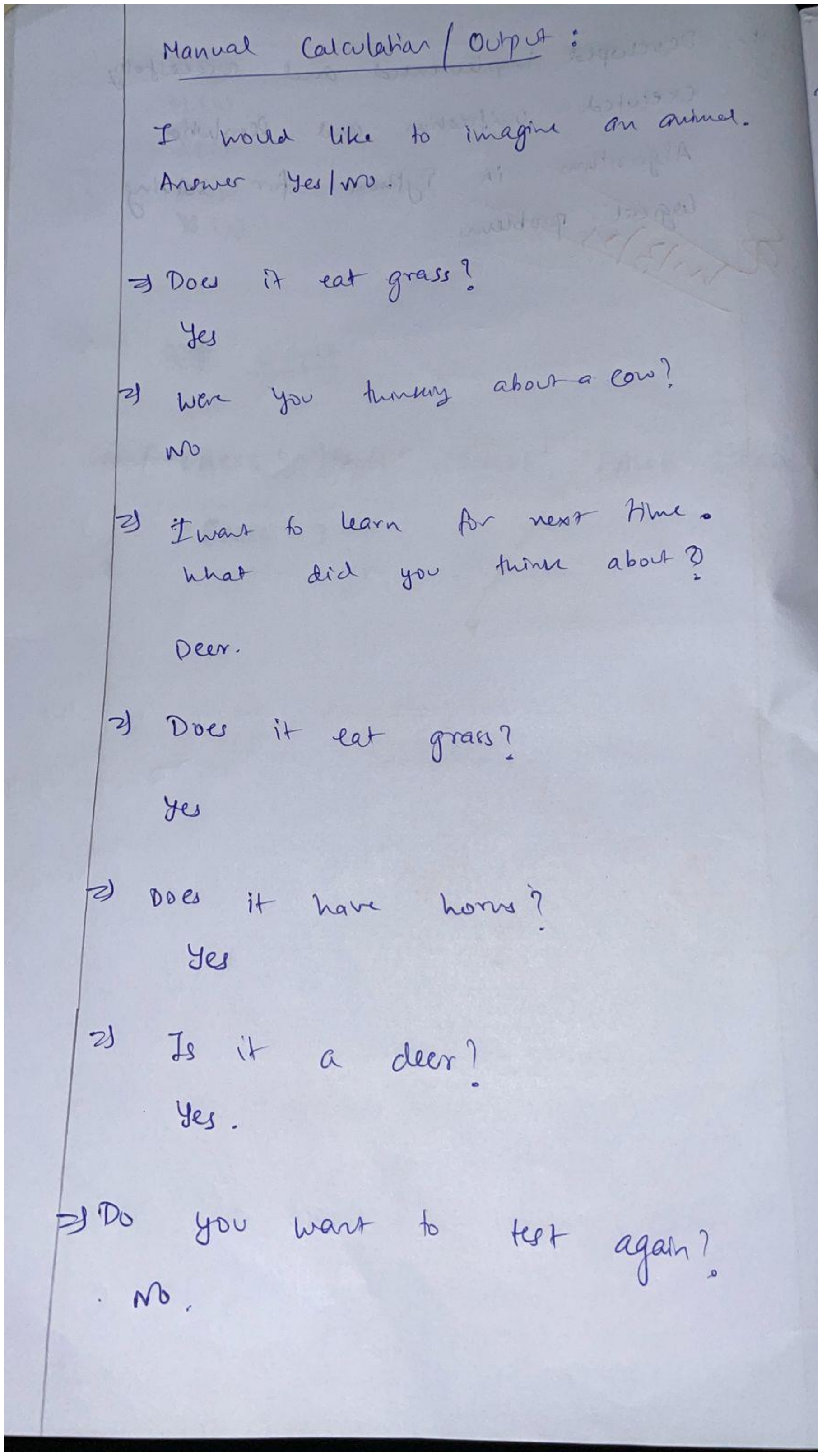
```
mammal    :- verify(has_hair), !.
mammal    :- verify(gives_milk).
bird      :- verify(has_feathers), !.
bird      :- verify(flys),
            verify(lays_eggs).
carnivore :- verify(eats_meat), !.
carnivore :- verify(has_pointed_teeth),
            verify(has_claws),
            verify(has_forward_eyes).
ungulate :- mammal,
           verify(has_hooves), !.
ungulate :- mammal,
           verify(chews_cud).
```

<u>PROGRAM(with OUTPUT)</u> :

```
import
sys

         def definiteNoun(s):
           s = s.lower().strip()
           if s in ['a', 'e', 'i', 'o', 'u', 'y']:
             return "an " + s
           else:
             return "a " + s

         def removeArticle(s):
           "Remove the definite article 'a' or 'an' from a noun."
           s = s.lower().strip()
           if s[0:3] == "an ": return s[3:]
           if s[0:2] == "a ": return s[2:]
           return s

         def makeQuestion(question, yes, no):
           return [question, yes, no]

         def isQuestion(p):
           "Check if node is a question (with answers), or a plain answer."
           return type(p).__name__ == "list"

         def askQuestion(question):
           print ("\r%s " % question,)
           return sys.stdin.readline().strip().lower()

         def getAnswer(question):
           if isQuestion(question):
             return askQuestion(question[0])
           else:
             return askQuestion("Were you thinking about %s?" % definiteNoun(question))

         def answeredYes(answer):
           if len(answer) > 0:
             return answer.lower()[0] == "y"
           return False

         def gameOver(message):
           global tries
```

```
  print ("")
  print ("\r%s" % message)
  print ("")

def playAgain():
  return answeredYes(askQuestion("Do you want to play again?"))

def correctGuess(message):
  global tries
  gameOver(message)

  if playAgain():
    print ("")
    tries = 0
    return Q
  else:
    sys.exit(0)

def nextQuestion(question, answer):
  global tries
  tries += 1

  if isQuestion(question):
    if answer:
      return question[1]
    else:
      return question[2]
  else:
    if answer:
      return correctGuess("I knew it!")
    else:
      return makeNewQuestion(question)

def replaceAnswer(tree, find, replace):
  if not isQuestion(tree):
    if tree == find:
      return replace
    else:
      return tree
  else:
    return makeQuestion(tree[0],
      replaceAnswer(tree[1], find, replace),
      replaceAnswer(tree[2], find, replace))

def makeNewQuestion(wrongAnimal):
  global Q, tries

  correctAnimal = removeArticle(askQuestion("I give up.  What did you think about?"))

  newQuestion = askQuestion("Enter a question that would distinguish %s from %s:"
      % (definiteNoun(correctAnimal), definiteNoun(wrongAnimal))).capitalize()

  yesAnswer = answeredYes(askQuestion("If I asked you this question " +
    "and you thought about %s, what would the correct answer be?" %
definiteNoun(correctAnimal)))

  # Create new question node
  if yesAnswer:
    q = makeQuestion(newQuestion, correctAnimal, wrongAnimal)
  else:
    q = makeQuestion(newQuestion, wrongAnimal, correctAnimal)
  Q = replaceAnswer(Q, wrongAnimal, q)
  tries = 0
  return Q

def addNewQuestion(wrongAnimal, newques, correct):
    global Q
    q = makeQuestion(newques, correct, wrongAnimal)
    Q = replaceAnswer(Q, wrongAnimal, q)
    return Q

tries = 0
Q = (makeQuestion('Does it have fur?', 'Tiger', 'Penguin'))
q = addNewQuestion('Tiger', 'Does it have dark spots?', 'Leopard')
q = addNewQuestion('Leopard', 'Is it the fastest animal?', 'Cheetah')
q = addNewQuestion('Penguin', 'Can it fly?', 'Parrot')
q = Q

print ("Imagine an animal.  I will try to guess which one.")
print ("You are only allowed to answer YES or NO.")
print ("")

try:
  while True:
    ans = answeredYes(getAnswer(q))
    q = nextQuestion(q, ans)
except KeyboardInterrupt:
  sys.exit(0)
except Exception:
  sys.exit(1)
```

Manual Calculations :



Manual Calculation / Output :

I would like to imagine an animal.
Answer Yes / no.

⇒ Does it eat grass?
Yes

⇒ Were you thinking about a cow?
no

⇒ I want to learn for next time.
what did you think about?

Deer.

⇒ Does it eat grass?
Yes

⇒ Does it have horns?
Yes

⇒ Is it a deer?
Yes.

⇒ Do you want to test again?
No.

<u>RESULT</u> :

Animal Detection problem successfully implemented using optimal artificial intelligence techniques under time complexity of O(n2).