106119029_lab8

November 3, 2021

0.1 Roll No. - 106119029

0.2 AI/ML Lab 8

Google Colab Link

0.3 Classes for all genralized things needed with a knowledgebase

1. class And => Used to represent the AND operation
2. class Or => Used to represent the OR Operation
3. Predicate => Use to represent a Single Rule

[1]:
```
from typing import List


class And:
    def __init__(self, lst: List[str]):
        self.symbols = lst

    def check(self, facts: List[str]):
        return all((x in facts) for x in self.symbols)

    def __str__(self):
        return '^'.join(self.symbols)


class Or:
    def __init__(self, lst: List[str]):
        self.symbols = lst

    def check(self, facts: List[str]):
        return any((x in facts) for x in self.symbols)

    def __str__(self):
        return '\\/'.join(self.symbols)


class Predicate:
    def __init__(self, s: str):
```

```
        lhs, rhs = [x.strip() for x in s.strip().split("=>")]
        self.lhs = And([lhs.split('^')[0].strip(), lhs.split('^')[1].strip()])
        self.rhs = rhs

    def __str__(self):
        return self.lhs.__str__() + "=>" + self.rhs.__str__()
```

0.4 Doing Forward Chaining

First we define the facts and clauses.

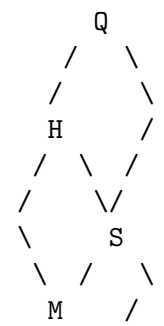
1. assert_fact => adds the fact to the fact list
2. forwardChaining => does the actual forwardChaining
3. printCountTable => used to print the countTable for the forward chaining on the given knowledgeBase

[2]:
```
is_changed = True

facts = ['A', 'B']
clauses = [Predicate(x) for x in ['H ^ S => Q', 'M ^ S => H',
           'H ^ A => M', 'M ^ B => S', 'A ^ B => M']]

needs_to_be_proved = 'Q'

def assert_fact(fact):
    global facts
    global is_changed
    if not (fact in facts):
        facts.append(fact)
        is_changed = True
    return is_changed


history = [facts.copy()]
print("Initial facts: {}".format(facts))

def forwardChaining():
  global facts
  global is_changed
  global history
  while is_changed:
      is_changed = False
      for clause in clauses:
          if clause.lhs.check(facts):
              if assert_fact(clause.rhs):
                  print("{} are all true and it implies {} is true and is added␣
 ↪to facts".format(
```

```
                    clause.lhs.symbols, clause.rhs))
                history.append(facts.copy())
                break

  print("Final facts: {}".format(facts))
  if needs_to_be_proved in facts:
      print("Q is in facts and hence proved")
  else:
      print("Q is not in facts and hence cant be proved")

def printCountTable():
  print("==========================================")
  print("Count table")
  print("==========================================")
  for fct in history:
      print(fct)

forwardChaining()
printCountTable()
```

```
Initial facts: ['A', 'B']
['A', 'B'] are all true and it implies M is true and is added to facts
['M', 'B'] are all true and it implies S is true and is added to facts
['M', 'S'] are all true and it implies H is true and is added to facts
['H', 'S'] are all true and it implies Q is true and is added to facts
Final facts: ['A', 'B', 'M', 'S', 'H', 'Q']
Q is in facts and hence proved
==========================================
Count table
==========================================
['A', 'B']
['A', 'B', 'M']
['A', 'B', 'M', 'S']
['A', 'B', 'M', 'S', 'H']
['A', 'B', 'M', 'S', 'H', 'Q']
```

0.5 And Or Tree



```
 /   \ /
A     B
```

0.6 Complexity and Completeness

0.6.1 Completeness

FC derives every atomic sentence that is entailed by KB 1. FC reaches a fixed point where no new atomic sentences are derived 2. Consider the final state as a model m, assigning true/false to symbols 3. Every clause in the original KB is true in m a1 … ak b 4. Hence m is a model of KB 5. If KB q, q is true in every model of KB, including m

0.6.2 Complexity

2^n (exponential)

[7]:
```
!cp drive/My Drive/Colab Notebooks/106119029_lab8.ipynb ./
```

```
cp: cannot stat 'drive/My': No such file or directory
cp: cannot stat 'Drive/Colab': No such file or directory
cp: cannot stat 'Notebooks/106119029_lab8.ipynb': No such file or directory
```

[ ]:
```

```