Artificial Intelligence
Lab Exp-6

SANDRA MARIA TONY
RA1911026010045
CSE-K1

UNIFICATION:

CODE:

```
def get_index_comma(string):
    index_list = list()
    par_count = 0

    for i in range(len(string)):
        if string[i] == ',' and par_count == 0:
            index_list.append(i)
        elif string[i] == '(':
            par_count += 1
        elif string[i] == ')':
            par_count -= 1

    return index_list



def is_variable(expr):
    for i in expr:
        if i == '(' or i == ')':
            return False
```

```
    return True
def process_expression(expr):
    expr = expr.replace(' ', '')
    index = None
    for i in range(len(expr)):
        if expr[i] == '(':
            index = i
            break
    predicate_symbol = expr[:index]
    expr = expr.replace(predicate_symbol, '')
    expr = expr[1:len(expr) - 1]
    arg_list = list()
    indices = get_index_comma(expr)

    if len(indices) == 0:
        arg_list.append(expr)
    else:
        arg_list.append(expr[:indices[0]])
        for i, j in zip(indices, indices[1:]):
            arg_list.append(expr[i + 1:j])
        arg_list.append(expr[indices[len(indices) - 1] + 1:])

    return predicate_symbol, arg_list


def get_arg_list(expr):
    _, arg_list = process_expression(expr)
```

RA1911026010045

```
    flag = True
    while flag:
        flag = False

        for i in arg_list:
            if not is_variable(i):
                flag = True
                _, tmp = process_expression(i)
                for j in tmp:
                    if j not in arg_list:
                        arg_list.append(j)
                arg_list.remove(i)

    return arg_list



def check_occurs(var, expr):
    arg_list = get_arg_list(expr)
    if var in arg_list:
        return True

    return False



def unify(expr1, expr2):

    if is_variable(expr1) and is_variable(expr2):
        if expr1 == expr2:
            return 'Null'
```

```
        else:
            return False
    elif is_variable(expr1) and not is_variable(expr2):
        if check_occurs(expr1, expr2):
            return False
        else:
            tmp = str(expr2) + '/' + str(expr1)
            return tmp
    elif not is_variable(expr1) and is_variable(expr2):
        if check_occurs(expr2, expr1):
            return False
        else:
            tmp = str(expr1) + '/' + str(expr2)
            return tmp
    else:
        predicate_symbol_1, arg_list_1 = process_expression(expr1)
        predicate_symbol_2, arg_list_2 = process_expression(expr2)

        # Step 2
        if predicate_symbol_1 != predicate_symbol_2:
            return False

        elif len(arg_list_1) != len(arg_list_2):
            return False
        else:

            sub_list = list()

            for i in range(len(arg_list_1)):
```

```
            tmp = unify(arg_list_1[i], arg_list_2[i])

            if not tmp:
                return False
            elif tmp == 'Null':
                pass
            else:
                if type(tmp) == list:
                    for j in tmp:
                        sub_list.append(j)
                else:
                    sub_list.append(tmp)

        # Step 6
        return sub_list


if __name__ == '__main__':

    f1 = 'P(b, f(y, b), g(x))'
    f2 = 'P(b, f(g(a), b), y)'
    # f1 = input('f1 : ')
    # f2 = input('f2 : ')

    result = unify(f1, f2)
    if not result:
        print('The process of Unification failed!')
    else:
        print('The process of Unification successful!')
```

print(result)

AWS CODE SCREENSHORT:

```
def get_index_comma(string):
    index_list = list()
    par_count = 0

    for i in range(len(string)):
        if string[i] == ',' and par_count == 0:
            index_list.append(i)
        elif string[i] == '(':
            par_count += 1
        elif string[i] == ')':
            par_count -= 1

    return index_list


def is_variable(expr):
    for i in expr:
        if i == '(' or i == ')':
            return False

    return True


def process_expression(expr):
    expr = expr.replace(' ', '')
    index = None
    for i in range(len(expr)):
        if expr[i] == '(':
```

```
                index = i
                break
        predicate_symbol = expr[:index]
        expr = expr.replace(predicate_symbol, '')
        expr = expr[1:len(expr) - 1]
        arg_list = list()
        indices = get_index_comma(expr)

        if len(indices) == 0:
            arg_list.append(expr)
        else:
            arg_list.append(expr[:indices[0]])
            for i, j in zip(indices, indices[1:]):
                arg_list.append(expr[i + 1:j])
            arg_list.append(expr[indices[len(indices) - 1] + 1:])

        return predicate_symbol, arg_list


def get_arg_list(expr):
    _, arg_list = process_expression(expr)

    flag = True
    while flag:
        flag = False

        for i in arg_list:
```

RA1911026010045

```
        for i in arg_list:
            if not is_variable(i):
                flag = True
                _, tmp = process_expression(i)
                for j in tmp:
                    if j not in arg_list:
                        arg_list.append(j)
                arg_list.remove(i)

    return arg_list


def check_occurs(var, expr):
    arg_list = get_arg_list(expr)
    if var in arg_list:
        return True

    return False


def unify(expr1, expr2):

    if is_variable(expr1) and is_variable(expr2):
        if expr1 == expr2:
            return 'Null'
        else:
            return False
```

RA1911026010045

```
81                  return False
82          elif is_variable(expr1) and not is_variable(expr2):
83              if check_occurs(expr1, expr2):
84                  return False
85              else:
86                  tmp = str(expr2) + '/' + str(expr1)
87                  return tmp
88          elif not is_variable(expr1) and is_variable(expr2):
89              if check_occurs(expr2, expr1):
90                  return False
91              else:
92                  tmp = str(expr1) + '/' + str(expr2)
93                  return tmp
94          else:
95              predicate_symbol_1, arg_list_1 = process_expression(expr1)
96              predicate_symbol_2, arg_list_2 = process_expression(expr2)
97
98              # Step 2
99              if predicate_symbol_1 != predicate_symbol_2:
100                 return False
101
102             elif len(arg_list_1) != len(arg_list_2):
103                 return False
104             else:
105
106                 sub_list = list()
107
108                 for i in range(len(arg_list_1)):
```

RA1911026010045

```
108             for i in range(len(arg_list_1)):
109                 tmp = unify(arg_list_1[i], arg_list_2[i])
110 
111                 if not tmp:
112                     return False
113                 elif tmp == 'Null':
114                     pass
115                 else:
116                     if type(tmp) == list:
117                         for j in tmp:
118                             sub_list.append(j)
119                     else:
120                         sub_list.append(tmp)
121 
122             # Step 6
123             return sub_list
124 
125 
126 if __name__ == '__main__':
127 
128     f1 = 'P(b, f(y, b), g(x))'
129     f2 = 'P(b, f(g(a), b), y)'
130     # f1 = input('f1 : ')
131     # f2 = input('f2 : ')
132 
133     result = unify(f1, f2)
134     if not result:
135         print('The process of Unification failed!')
```

```
130     # f1 = input('f1 : ')
131     # f2 = input('f2 : ')
132 
133     result = unify(f1, f2)
134     if not result:
135         print('The process of Unification failed!')
136     else:
137         print('The process of Unification successful!')
138         print(result)
139 
140 
```

RA1911026010045

OUTPUT:

RA1911026010045

RESOLUTION:

```
import copy

import time




class Parameter:

    variable_count = 1



    def __init__(self, name=None):

        if name:

            self.type = "Constant"

            self.name = name

        else:

            self.type = "Variable"

            self.name = "v" + str(Parameter.variable_count)

            Parameter.variable_count += 1



    def isConstant(self):

        return self.type == "Constant"
```

```
    def unify(self, type_, name):
        self.type = type_
        self.name = name

    def __eq__(self, other):
        return self.name == other.name

    def __str__(self):
        return self.name


class Predicate:
    def __init__(self, name, params):
        self.name = name
        self.params = params

    def __eq__(self, other):
        return self.name == other.name and all(a == b for a, b in zip(self.params, other.params))
```

```
    def __str__(self):
        return self.name + "(" + ",".join(str(x) for x in self.params) + ")"

    def getNegatedPredicate(self):
        return Predicate(negatePredicate(self.name), self.params)


class Sentence:
    sentence_count = 0

    def __init__(self, string):
        self.sentence_index = Sentence.sentence_count
        Sentence.sentence_count += 1
        self.predicates = []
        self.variable_map = {}
        local = {}

        for predicate in string.split("|"):
            name = predicate[:predicate.find("(")]
            params = []
```

```
        for param in predicate[predicate.find("(") + 1: predicate.find(")")].split(","):
            if param[0].islower():
                if param not in local:  # Variable
                    local[param] = Parameter()
                    self.variable_map[local[param].name] = local[param]
                new_param = local[param]
            else:
                new_param = Parameter(param)
                self.variable_map[param] = new_param

            params.append(new_param)

        self.predicates.append(Predicate(name, params))

    def getPredicates(self):
        return [predicate.name for predicate in self.predicates]

    def findPredicates(self, name):
        return [predicate for predicate in self.predicates if predicate.name == name]
```

```
    def removePredicate(self, predicate):
        self.predicates.remove(predicate)
        for key, val in self.variable_map.items():
            if not val:
                self.variable_map.pop(key)

    def containsVariable(self):
        return any(not param.isConstant() for param in self.variable_map.values())

    def __eq__(self, other):
        if len(self.predicates) == 1 and self.predicates[0] == other:
            return True
        return False

    def __str__(self):
        return "".join([str(predicate) for predicate in self.predicates])


class KB:
```

```
    def __init__(self, inputSentences):
        self.inputSentences = [x.replace(" ", "") for x in inputSentences]
        self.sentences = []
        self.sentence_map = {}


    def prepareKB(self):
        self.convertSentencesToCNF()
        for sentence_string in self.inputSentences:
            sentence = Sentence(sentence_string)
            for predicate in sentence.getPredicates():
                self.sentence_map[predicate] = self.sentence_map.get(
                    predicate, []) + [sentence]


    def convertSentencesToCNF(self):
        for sentenceIdx in range(len(self.inputSentences)):
            # Do negation of the Premise and add them as literal
            if "=>" in self.inputSentences[sentenceIdx]:
                self.inputSentences[sentenceIdx] = negateAntecedent(
                    self.inputSentences[sentenceIdx])
```

```
    def askQueries(self, queryList):
        results = []

        for query in queryList:
            negatedQuery = Sentence(negatePredicate(query.replace(" ", "")))
            negatedPredicate = negatedQuery.predicates[0]
            prev_sentence_map = copy.deepcopy(self.sentence_map)
            self.sentence_map[negatedPredicate.name] = self.sentence_map.get(
                negatedPredicate.name, []) + [negatedQuery]
            self.timeLimit = time.time() + 40

            try:
                result = self.resolve([negatedPredicate], [
                                      False]*(len(self.inputSentences) + 1))
            except:
                result = False

            self.sentence_map = prev_sentence_map

            if result:
```

```
            results.append("TRUE")
        else:
            results.append("FALSE")

    return results

def resolve(self, queryStack, visited, depth=0):
    if time.time() > self.timeLimit:
        raise Exception
    if queryStack:
        query = queryStack.pop(-1)
        negatedQuery = query.getNegatedPredicate()
        queryPredicateName = negatedQuery.name
        if queryPredicateName not in self.sentence_map:
            return False
        else:
            queryPredicate = negatedQuery
            for kb_sentence in self.sentence_map[queryPredicateName]:
                if not visited[kb_sentence.sentence_index]:
                    for kbPredicate in kb_sentence.findPredicates(queryPredicateName):
```

```
canUnify, substitution = performUnification(
    copy.deepcopy(queryPredicate), copy.deepcopy(kbPredicate))

if canUnify:
    newSentence = copy.deepcopy(kb_sentence)
    newSentence.removePredicate(kbPredicate)
    newQueryStack = copy.deepcopy(queryStack)

    if substitution:
        for old, new in substitution.items():
            if old in newSentence.variable_map:
                parameter = newSentence.variable_map[old]
                newSentence.variable_map.pop(old)
                parameter.unify(
                    "Variable" if new[0].islower() else "Constant", new)
                newSentence.variable_map[new] = parameter

        for predicate in newQueryStack:
            for index, param in enumerate(predicate.params):
```

```
                        if param.name in substitution:
                            new = substitution[param.name]
                            predicate.params[index].unify(
                                "Variable" if new[0].islower() else "Constant", new)

                for predicate in newSentence.predicates:
                    newQueryStack.append(predicate)

                new_visited = copy.deepcopy(visited)
                if kb_sentence.containsVariable() and len(kb_sentence.predicates) > 1:
                    new_visited[kb_sentence.sentence_index] = True

                if self.resolve(newQueryStack, new_visited, depth + 1):
                    return True
            return False
        return True


def performUnification(queryPredicate, kbPredicate):
    substitution = {}
```

```
    if queryPredicate == kbPredicate:
        return True, {}
    else:
        for query, kb in zip(queryPredicate.params, kbPredicate.params):
            if query == kb:
                continue
            if kb.isConstant():
                if not query.isConstant():
                    if query.name not in substitution:
                        substitution[query.name] = kb.name
                    elif substitution[query.name] != kb.name:
                        return False, {}
                    query.unify("Constant", kb.name)
                else:
                    return False, {}
            else:
                if not query.isConstant():
                    if kb.name not in substitution:
                        substitution[kb.name] = query.name
                    elif substitution[kb.name] != query.name:
```

```
                return False, {}
            kb.unify("Variable", query.name)
        else:
            if kb.name not in substitution:
                substitution[kb.name] = query.name
            elif substitution[kb.name] != query.name:
                return False, {}
    return True, substitution


def negatePredicate(predicate):
    return predicate[1:] if predicate[0] == "~" else "~" + predicate


def negateAntecedent(sentence):
    antecedent = sentence[:sentence.find("=>")]
    premise = []

    for predicate in antecedent.split("&"):
        premise.append(negatePredicate(predicate))
```

```
    premise.append(sentence[sentence.find("=>") + 2:])
    return "|".join(premise)



def getInput(filename):
    with open(filename, "r") as file:
        noOfQueries = int(file.readline().strip())
        inputQueries = [file.readline().strip() for _ in range(noOfQueries)]
        noOfSentences = int(file.readline().strip())
        inputSentences = [file.readline().strip()
                          for _ in range(noOfSentences)]
        return inputQueries, inputSentences



def printOutput(filename, results):
    print(results)
    with open(filename, "w") as file:
        for line in results:
            file.write(line)
```

```
        file.write("\n")
    file.close()



if __name__ == '__main__':
                inputQueries_, inputSentences_ = getInput(r"C:\Users\Hetu Prakash Patel\Desktop\input.txt")
    knowledgeBase = KB(inputSentences_)
    knowledgeBase.prepareKB()
    results_ = knowledgeBase.askQueries(inputQueries_)
    printOutput("output.txt", results_)
```

AWS CODE SCREENSHORT:

```
1  import copy
2  import time
3
4
5  class Parameter:
6      variable_count = 1
7
8      def __init__(self, name=None):
9          if name:
10             self.type = "Constant"
11             self.name = name
12         else:
13             self.type = "Variable"
14             self.name = "v" + str(Parameter.variable_count)
15             Parameter.variable_count += 1
16
17     def isConstant(self):
18         return self.type == "Constant"
19
20     def unify(self, type_, name):
21         self.type = type_
22         self.name = name
23
24     def __eq__(self, other):
25         return self.name == other.name
26
27     def __str__(self):
28         return self.name
```

RA1911026010045

```
class Predicate:
    def __init__(self, name, params):
        self.name = name
        self.params = params

    def __eq__(self, other):
        return self.name == other.name and all(a == b for a, b in zip(self.params, other.params))

    def __str__(self):
        return self.name + "(" + ",".join(str(x) for x in self.params) + ")"

    def getNegatedPredicate(self):
        return Predicate(negatePredicate(self.name), self.params)


class Sentence:
    sentence_count = 0

    def __init__(self, string):
        self.sentence_index = Sentence.sentence_count
        Sentence.sentence_count += 1
        self.predicates = []
        self.variable_map = {}
        local = {}

        for predicate in string.split("|"):
            name = predicate[:predicate.find("(")]
            params = []
```

1:1

```
                new_param = Parameter(param)
                self.variable_map[param] = new_param

            params.append(new_param)

        self.predicates.append(Predicate(name, params))

    def getPredicates(self):
        return [predicate.name for predicate in self.predicates]

    def findPredicates(self, name):
        return [predicate for predicate in self.predicates if predicate.name == name]

    def removePredicate(self, predicate):
        self.predicates.remove(predicate)
        for key, val in self.variable_map.items():
            if not val:
                self.variable_map.pop(key)

    def containsVariable(self):
        return any(not param.isConstant() for param in self.variable_map.values())

    def __eq__(self, other):
        if len(self.predicates) == 1 and self.predicates[0] == other:
            return True
        return False

    def __str__(self):
```

1:1 Pyth

RA1911026010045

```
127                 negatedPredicate.name, []) + [negatedQuery]
128             self.timeLimit = time.time() + 40
129 
130             try:
131                 result = self.resolve([negatedPredicate], [
132                                       False]*(len(self.inputSentences) + 1))
133             except:
134                 result = False
135 
136             self.sentence_map = prev_sentence_map
137 
138             if result:
139                 results.append("TRUE")
140             else:
141                 results.append("FALSE")
142 
143         return results
144 
145     def resolve(self, queryStack, visited, depth=0):
146         if time.time() > self.timeLimit:
147             raise Exception
148         if queryStack:
149             query = queryStack.pop(-1)
150             negatedQuery = query.getNegatedPredicate()
151             queryPredicateName = negatedQuery.name
152             if queryPredicateName not in self.sentence_map:
153                 return False
154             else:
```

RA1911026010045

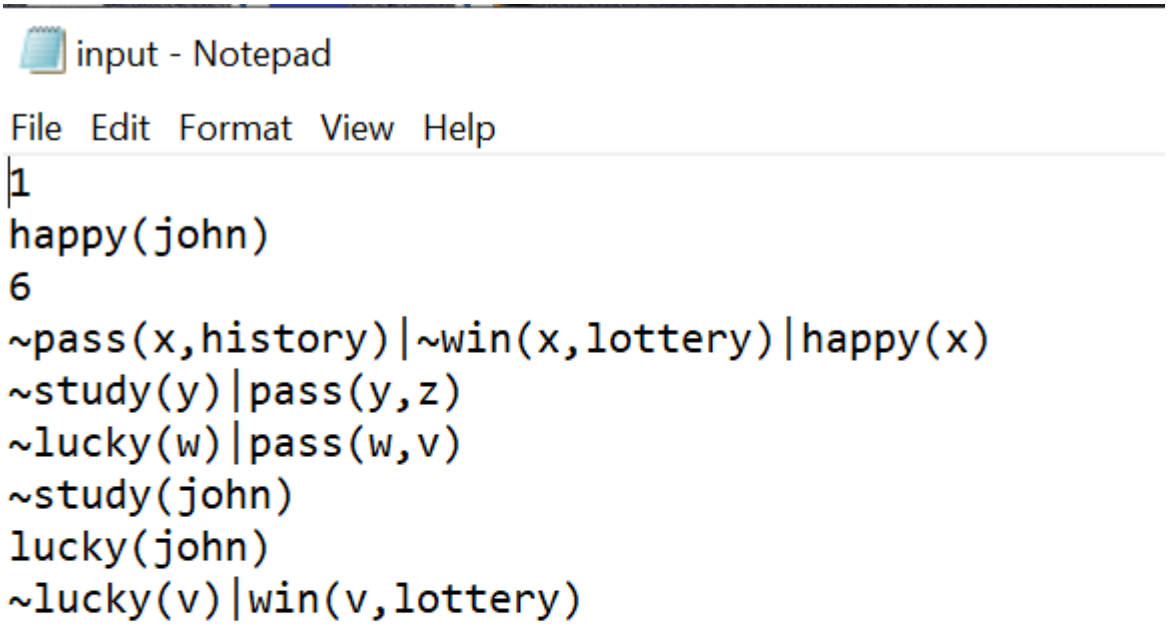
```
151             queryPredicateName = negatedQuery.name
152             if queryPredicateName not in self.sentence_map:
153                 return False
154             else:
155                 queryPredicate = negatedQuery
156                 for kb_sentence in self.sentence_map[queryPredicateName]:
157                     if not visited[kb_sentence.sentence_index]:
158                         for kbPredicate in kb_sentence.findPredicates(queryPredicateName):
159
160                             canUnify, substitution = performUnification(
161                                 copy.deepcopy(queryPredicate), copy.deepcopy(kbPredicate))
162
163                             if canUnify:
164                                 newSentence = copy.deepcopy(kb_sentence)
165                                 newSentence.removePredicate(kbPredicate)
166                                 newQueryStack = copy.deepcopy(queryStack)
167
168                                 if substitution:
169                                     for old, new in substitution.items():
170                                         if old in newSentence.variable_map:
171                                             parameter = newSentence.variable_map[old]
172                                             newSentence.variable_map.pop(old)
173                                             parameter.unify(
174                                                 "Variable" if new[0].islower() else "Constant", new)
175                                             newSentence.variable_map[new] = parameter
176
177                                 for predicate in newQueryStack:
178                                     for index, param in enumerate(predicate.params):
```

1:1 Py

```
241     return "|".join(premise)
242
243
244 def getInput(filename):
245     with open(filename, "r") as file:
246         noOfQueries = int(file.readline().strip())
247         inputQueries = [file.readline().strip() for _ in range(noOfQueries)]
248         noOfSentences = int(file.readline().strip())
249         inputSentences = [file.readline().strip()
250                           for _ in range(noOfSentences)]
251         return inputQueries, inputSentences
252
253
254 def printOutput(filename, results):
255     print(results)
256     with open(filename, "w") as file:
257         for line in results:
258             file.write(line)
259             file.write("\n")
260     file.close()
261
262
263 if __name__ == '__main__':
264     inputQueries_, inputSentences_ = getInput(r"C:\Users\An\Desktop\input.text")
265     knowledgeBase = KB(inputSentences_)
266     knowledgeBase.prepareKB()
267     results_ = knowledgeBase.askQueries(inputQueries_)
268     printOutput("output.txt", results_)
```

045_input - Notepad

Input.txt:



045_output.txt