Mata Kuliah	**:**	**Kecerdasan Buatan**
Kelas	**:**	TI-2H
NIM	**:**	**2041720074**
Nama	**:**	**Mochammad Hairullah**

Praktikum

1. Implementasikan kode program diatas

   Input.txt

   1

   B(John,Bob)

   13

   A(x,y) => B(x,y)

   G(x,y) => B(x,y)

   C(c,d) => A(c,d)

   B(x,y) => G(x,y)

   F(y) => G(x,y)

   G(x,y) ^ F(y) => H(x,y)

   I(z) => F(z)

   F(z) => I(z)

   C(Jie,Joe)

   C(Melissa,Mary)

   G(John,Bob)

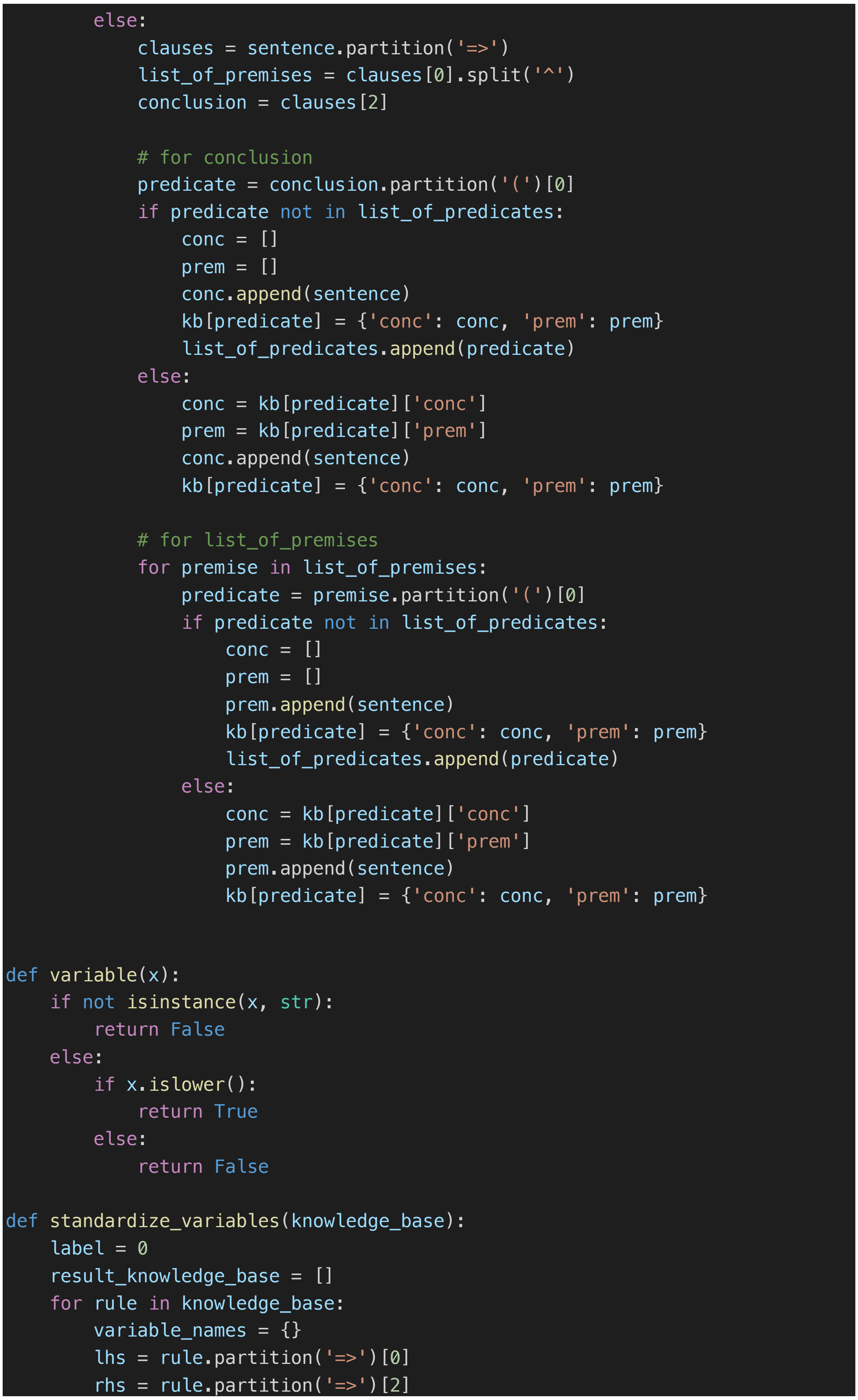
   I(Amy)

   F(Bob)

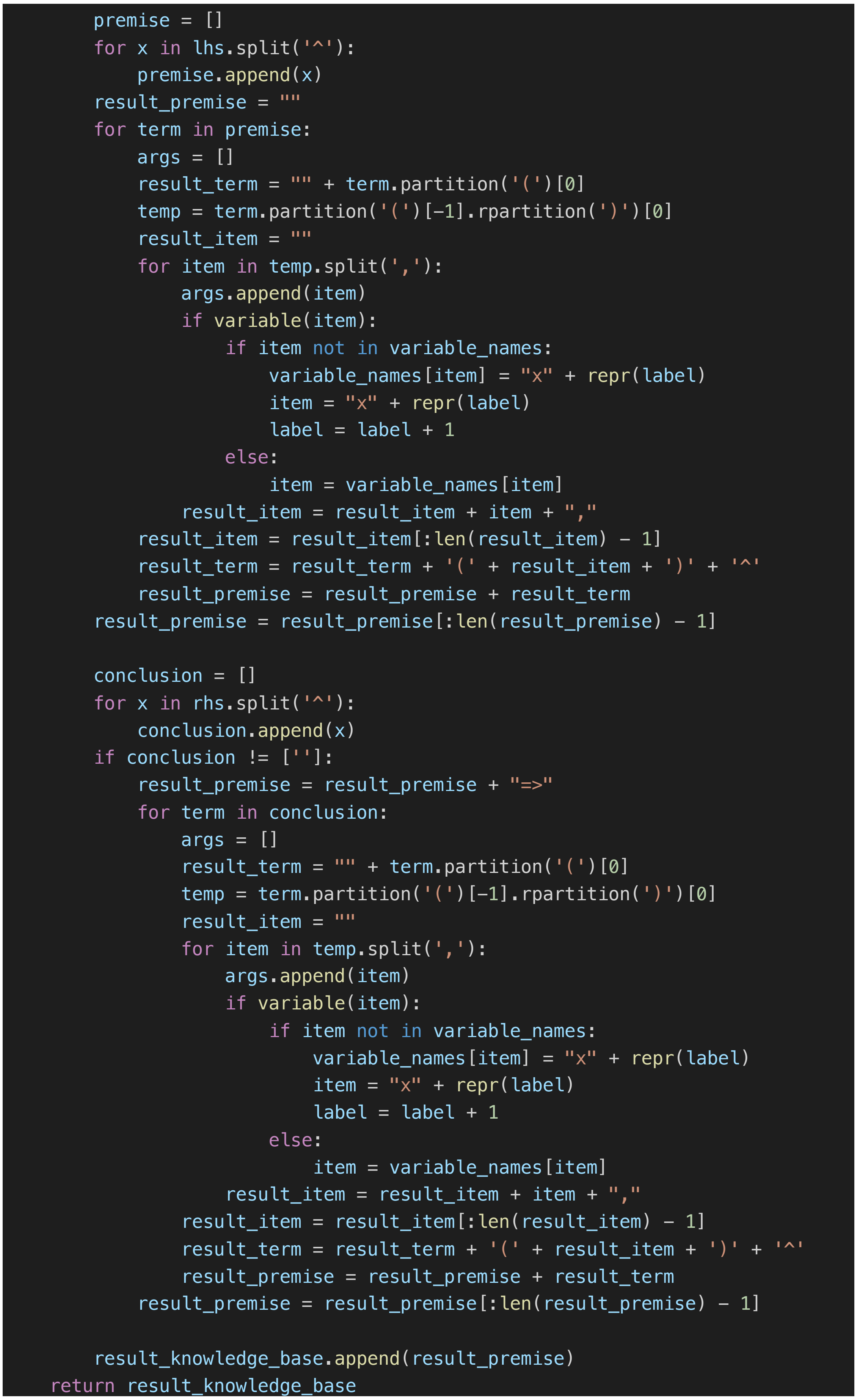
   BackwardChaining.py

```python
# Mochammad Hairullah
# nim 2041720074

from copy import deepcopy

kb = {}
list_of_predicates = []
list_of_explored_rules = []

def fetch_rules(goal):
    global kb
    global list_of_predicates

    print("fetch_rules for goal:- ", goal)
    list_of_rules = []
    predicate = goal.partition('(')[0]
    print("\t", predicate, kb[predicate]['conc'])
    list_of_rules = list_of_rules + kb[predicate]['conc']
    return list_of_rules


def subst(theta, first):
    print("\tsubst: ", theta, first)
    predicate = first.partition('(')[0]
    list = (first.partition('(')[-1].rpartition(')')[0]).split(',')
    print("\t", list)
    for i in range(len(list)):
        if variable(list[i]):
            if list[i] in theta:
                list[i] = theta[list[i]]
    print("\t", predicate + '(' + ','.join(list) + ')')
    return predicate + '(' + ','.join(list) + ')'


def variable(x):
    if not isinstance(x, str):
        return False
    else:
        if x.islower():
            return True
        else:
            return False


def compound(x):
    if not isinstance(x, str):
        return False
    else:
        if '(' in x and ')' in x:
            return True
        else:
            return False


def list(x):
    if not isinstance(x, str):
        return True
    else:
        return False


def unify_var(var, x, theta):
    print("IN unify_var", var, x, theta)
    if var in theta:
        print("var in theta", var, theta)
        return unify(theta[var], x, theta)
    elif x in theta:
        print("x in theta", x, theta)
        return unify(var, theta[x], theta)
    else:
        theta[var] = x
        print("not in theta", theta[var])
        return theta


def check_theta(theta):
    for entry in theta:
        if variable(theta[entry]):
            if theta[entry] in theta:
                print("in check_theta. theta changed")
                theta[entry] = theta[theta[entry]]
    return theta


def unify(x, y, theta):
    print("\tunify", x, y, theta)
    if theta == None:
        print("\tin theta is None")
        return None
    elif x == y:
        print("\tin x=y")
        return check_theta(theta)
    elif variable(x) is True:
        print("\tin variable(x)")
        return unify_var(x, y, theta)
    elif variable(y) is True:
        print("\tin variable(y)")
        return unify_var(y, x, theta)
    elif compound(x) and compound(y):
        print("\tin compound")
        x_args = []
        temp = x.partition('(')[-1].rpartition(')')[0]
        for item in temp.split(','):
            x_args.append(item)
        y_args = []
        temp = y.partition('(')[-1].rpartition(')')[0]
        for item in temp.split(','):
            y_args.append(item)
        x_op = x.partition('(')[0]
        y_op = y.partition('(')[0]
        return unify(x_args, y_args, unify(x_op, y_op, theta))
    elif list(x) and list(y):
        print("\tin list")
        return unify(x[1:], y[1:], unify(x[0], y[0], theta))
    else:
        print("\tin else")
        return None
```

```python
def fol_bc_ask(query, theta):
    global kb
    global list_of_predicates
    global list_of_explored_rules

    print("Backward Chaining")
    list_of_rules = fetch_rules(query)
    for rule in list_of_rules:
        print("taken RULE", rule)
        list_of_explored_rules = []
        list_of_explored_rules.append(query)
        print("\t",query, "added to list_of_explored_rules")
        lhs = rule.partition('=>')[0]
        rhs = rule.partition('=>')[2]
        print("lhs: ", lhs, " rhs: ", rhs)
        print("theta in rule", theta)
        theta1 = unify(rhs, query, theta)
        if theta1 != None:
            list_of_premises = lhs.split('^')
            print("list_of_premises: ", list_of_premises)
            theta2 = fol_bc_and(theta1, list_of_premises)
            if theta2 != None:
                return theta2

    print("None of the rules worked out", query)
    return None


def fol_bc_and(theta, list_of_premises):
    global kb
    global list_of_predicates

    print("\tand: ", list_of_premises)
    print("\ttheta: ", theta)
    if theta == None:
        return None
    else:
        if list_of_premises != []:
            temp_list = []
            for each_premise in list_of_premises:
                temp = subst(theta, each_premise)
                temp_list.append(temp)
            list_of_premises = temp_list
            first_premise = list_of_premises[0]
            rest_premise = list_of_premises[1:]
            subs = list_of_premises[0]
            if subs != '()':
                if subs in list_of_explored_rules:
                    print(subs, " already in list_of_explored_rules")
                    return None
                else:
                    print(subs, " added to list_of_explored_rules")
                    list_of_explored_rules.append(subs)
                theta = fol_bc_or_sub(subs, {}, rest_premise)
            else:
                return theta
        return theta


def fol_bc_or_sub(query, theta, rest):
    global kb
    global list_of_predicates

    print("\tOR sub")
    list_of_rules = fetch_rules(query)
    print("\tLIST_OF_RULES", list_of_rules)
    for rule in list_of_rules:
        print("\tRULE", rule)
        lhs = rule.partition('=>')[0]
        rhs = rule.partition('=>')[2]
        print("\n\tlhs: ", lhs, " rhs: ", rhs)
        print("\ntheta in rule", theta)
        theta1 = unify(rhs, query, deepcopy(theta))
        if theta1 != None:
            list_of_premises = lhs.split('^')
            print("\tlist_of_premises: ", list_of_premises)
            theta2 = fol_bc_and(theta1, list_of_premises)
            theta3 = fol_bc_and(theta2, rest)
            if theta3 != None:
                return theta3

    print("\tNone of the rules worked out", query)
    return None


def add_to_kb(knowledge_base):
    global kb
    global list_of_predicates

    for sentence in knowledge_base:
        if '=>' not in sentence:
            predicate = sentence.partition('(')[0]
            if predicate not in list_of_predicates:
                conc = []
                prem = []
                conc.append("=>" + sentence)
                kb[predicate] = {'conc': conc, 'prem': prem}
                list_of_predicates.append(predicate)
            else:
                conc = kb[predicate]['conc']
                prem = kb[predicate]['prem']
                conc.append("=>" + sentence)
                kb[predicate] = {'conc': conc, 'prem': prem}
```



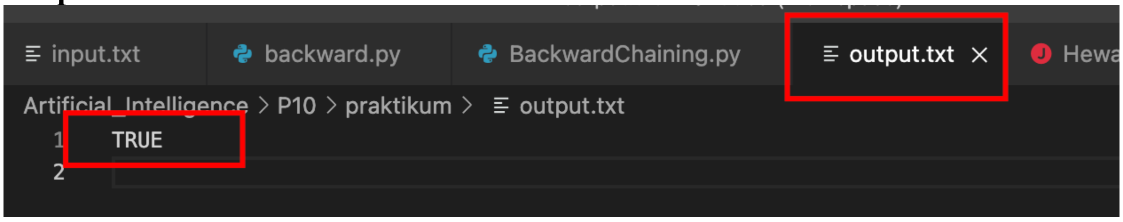
```
        else:
            clauses = sentence.partition('=>')
            list_of_premises = clauses[0].split('^')
            conclusion = clauses[2]

            # for conclusion
            predicate = conclusion.partition('(')[0]
            if predicate not in list_of_predicates:
                conc = []
                prem = []
                conc.append(sentence)
                kb[predicate] = {'conc': conc, 'prem': prem}
                list_of_predicates.append(predicate)
            else:
                conc = kb[predicate]['conc']
                prem = kb[predicate]['prem']
                conc.append(sentence)
                kb[predicate] = {'conc': conc, 'prem': prem}

            # for list_of_premises
            for premise in list_of_premises:
                predicate = premise.partition('(')[0]
                if predicate not in list_of_predicates:
                    conc = []
                    prem = []
                    prem.append(sentence)
                    kb[predicate] = {'conc': conc, 'prem': prem}
                    list_of_predicates.append(predicate)
                else:
                    conc = kb[predicate]['conc']
                    prem = kb[predicate]['prem']
                    prem.append(sentence)
                    kb[predicate] = {'conc': conc, 'prem': prem}


def variable(x):
    if not isinstance(x, str):
        return False
    else:
        if x.islower():
            return True
        else:
            return False

def standardize_variables(knowledge_base):
    label = 0
    result_knowledge_base = []
    for rule in knowledge_base:
        variable_names = {}
        lhs = rule.partition('=>')[0]
        rhs = rule.partition('=>')[2]
```



```python
        premise = []
        for x in lhs.split('^'):
            premise.append(x)
        result_premise = ""
        for term in premise:
            args = []
            result_term = "" + term.partition('(')[0]
            temp = term.partition('(')[-1].rpartition(')')[0]
            result_item = ""
            for item in temp.split(','):
                args.append(item)
                if variable(item):
                    if item not in variable_names:
                        variable_names[item] = "x" + repr(label)
                        item = "x" + repr(label)
                        label = label + 1
                    else:
                        item = variable_names[item]
                result_item = result_item + item + ","
            result_item = result_item[:len(result_item) - 1]
            result_term = result_term + '(' + result_item + ')' + '^'
            result_premise = result_premise + result_term
        result_premise = result_premise[:len(result_premise) - 1]

        conclusion = []
        for x in rhs.split('^'):
            conclusion.append(x)
        if conclusion != ['']:
            result_premise = result_premise + "=>"
            for term in conclusion:
                args = []
                result_term = "" + term.partition('(')[0]
                temp = term.partition('(')[-1].rpartition(')')[0]
                result_item = ""
                for item in temp.split(','):
                    args.append(item)
                    if variable(item):
                        if item not in variable_names:
                            variable_names[item] = "x" + repr(label)
                            item = "x" + repr(label)
                            label = label + 1
                        else:
                            item = variable_names[item]
                    result_item = result_item + item + ","
                result_item = result_item[:len(result_item) - 1]
                result_term = result_term + '(' + result_item + ')' + '^'
                result_premise = result_premise + result_term
            result_premise = result_premise[:len(result_premise) - 1]

        result_knowledge_base.append(result_premise)
    return result_knowledge_base
```

```
#Main

fn="input.txt"
queries = []
knowledge_base = []
f1=open(fn, "r")
input = f1.readlines()
input = [x.strip() for x in input]

for i in range(1, int(input[0]) + 1):
    queries.append(input[i].replace(" ", ""))
for i in range(int(input[0]) + 2, int(input[int(input[0]) + 1]) +
int(input[0]) + 2):
    knowledge_base.append(input[i].replace(" ", ""))
knowledge_base = standardize_variables(knowledge_base)

kb = {}
list_of_predicates = []
add_to_kb(knowledge_base)

fileOut = open("output.txt", "w")
for query in queries:
    result = fol_bc_ask(query, {})
    if result != None:
        print("True", result)
        fileOut.write("TRUE" + "\n")
    else:
        print("False", result)
        fileOut.write("FALSE" + "\n")

fileOut.close()
f1.close
```

Output



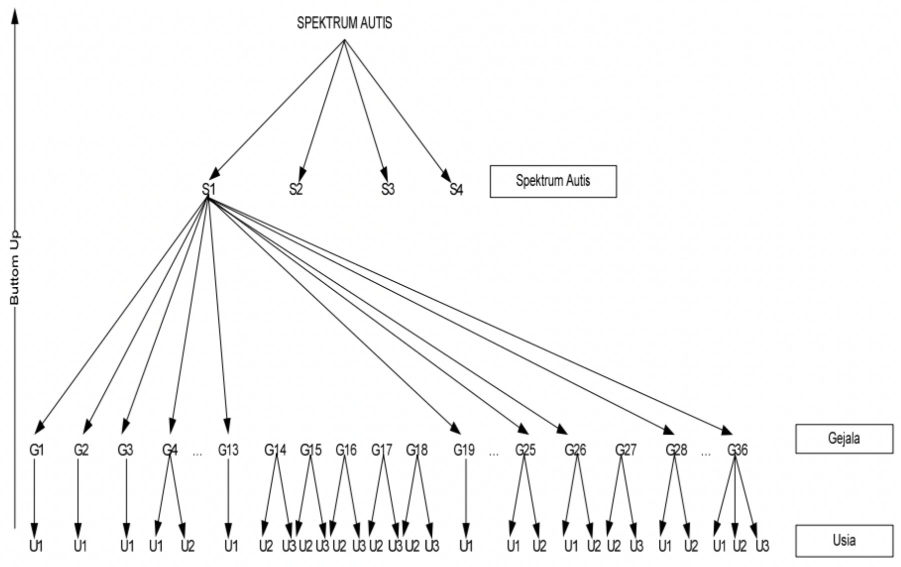
```
TRUE

```

2. Lakukan review terhadap jurnal yang diberikan
   a. Latar belakang – tuliskan masalah

      Salah satu masalah yang ada pada seseorang atau gangguan dalam tumbuh kembang yang sering terjadi belakangan ini adalah *Autisme* yaitu ketidaknormalan perkembangan mental sehingga menyebabkan anak sulit untuk berinteraksi sosial.

   b. Bagaimana forward chaingingnya

- Mencari fakta fakta yang akan dijadikan kesimpulan
- Yang dibutuhkan adalah table Spektrum Autisme, table usia, table gejala dan table terapi untuk membuat sebuah prediksi forward chaining
- S1 adalah spektrum autis infantile padausia 0-1 tahun
  - Bayi tampak tenang dan jarang menangis (G1), Sulit bila digendong (G2)
  - Tidak mengoceh (G3), Tidak senang diayun di lutut (G4), Tidak mau menatap mata (G13)
  - Perkembangan agak terlambat misal dalam berjalan (G19)
  - Menolak untuk dipeluk (G25), Suka tiba-tiba menangis dan tertawa tanpa sebab (G26)
  - Bermain dengan benda yang bukan mainan missal ujung selimut (G28)
  - Tidak ada senyum sosial saat bertemu orang lain (G36)
- Lalu Setiap spektrum di atas akan dibuat kombinasi untuk setiap kemungkinan sampai 80% gejala terpenuhi.


SPEKTRUM AUTIS
S1
S2
S3
S4
Spektrum Autis
Buttom Up
G1 G2 G3 G4 ... G13 G14 G15 G16 G17 G18 G19 ... G25 G26 G27 G28 ... G36
Gejala
U1 U1 U1 U1 U2 U1 U2 U3 U2 U3 U2 U3 U2 U3 U2 U3 U1 U1 U2 U1 U2 U2 U3 U1 U2 U1 U2 U3
Usia

Gambar 4. Penelusuran dengan *bottom up reasoning*

- Sistem akan menampung inputan setelah menjawab pertanyaan pada fakta fakta yang ada pada pertanyaan
- Lalu akan di cek rule berdasarkan inputan yang ditampung jika
- ditemukan ulangi langkah 1 sampai dengan langkah 3. Jika tidak ditemukan maka berikan default output. Lalu memberikan solusinya

- contoh rule dan kode programnya sesuai aturan yang dipakai pada spektrum autis

```
def logika(spektrum, usia, gejala):
    gejalaArray =
['G1','G2','G3','G4','G13','G19','G25','G26','G28','G36']
    terapi = ['T1']
    if spektrum == 'S1' :
        if usia == 'U1' :
            if gejala == gejalaArray : # jika sudah terpenuhi semua
maka akan menghasilkan sebuah keputusan
                print("Terapi yang dapat dilakukan", terapi)
    return terapi
```