中山大学计算机学院人工智能本科生实验报告（2022学年春季学期)

课程名称：Artificial Intelligence

教学班级:2班 专业（方向）：计算机科学与技术

学号：21307174 姓名：刘俊杰

一、实验题目

归结原理

二、实验内容

- 编写程序，实现一阶逻辑归结算法，并用于求解给出的三个逻辑推理问题，要求输出按照如下格式：

```
1. (P(x),Q(g(x)))
2. (R(a),Q(z),?P(a))
3. R[1a,2c]{x=a} (Q(g(a)),R(a),Q(z))
```

- “R” 表示归结步骤.
- “1a” 表示第一个子句(1-th)中的第一个 (a-th)个原子公式，即P(x).
- “2c”表示第二个子句(1-th)中的第三个 (c-th)个原子公式，即?P(a).
- “1a”和“2c”是冲突的，所以应用最小合一{x = a}.

1. 试验任务一：Alpine Club

```
A(tony)
A(mike)
A(john)
L(tony, rain)
L(tony, snow)
(?A(x), S(x), C(x))
(?C(y), ?L(y, rain))
(L(z, snow), ?S(z))
(?L(tony, u), ?L(mike, u))
(L(tony, v), L(mike, v))
(?A(w), ?C(w), S(w))
```

2. 试验任务二：Graduate Student

```
GradStudent(sue)
(?GradStudent(x), Student(x))
```

```
(?Student(x), HardWorker(x))
?HardWorker(sue)
```

3. 试验任务三：Block World

```
On(aa,bb)
On(bb,cc)
Green(aa)
?Green(cc)
(?On(x,y), ?Green(x), Green(y))
```

1. 算法原理

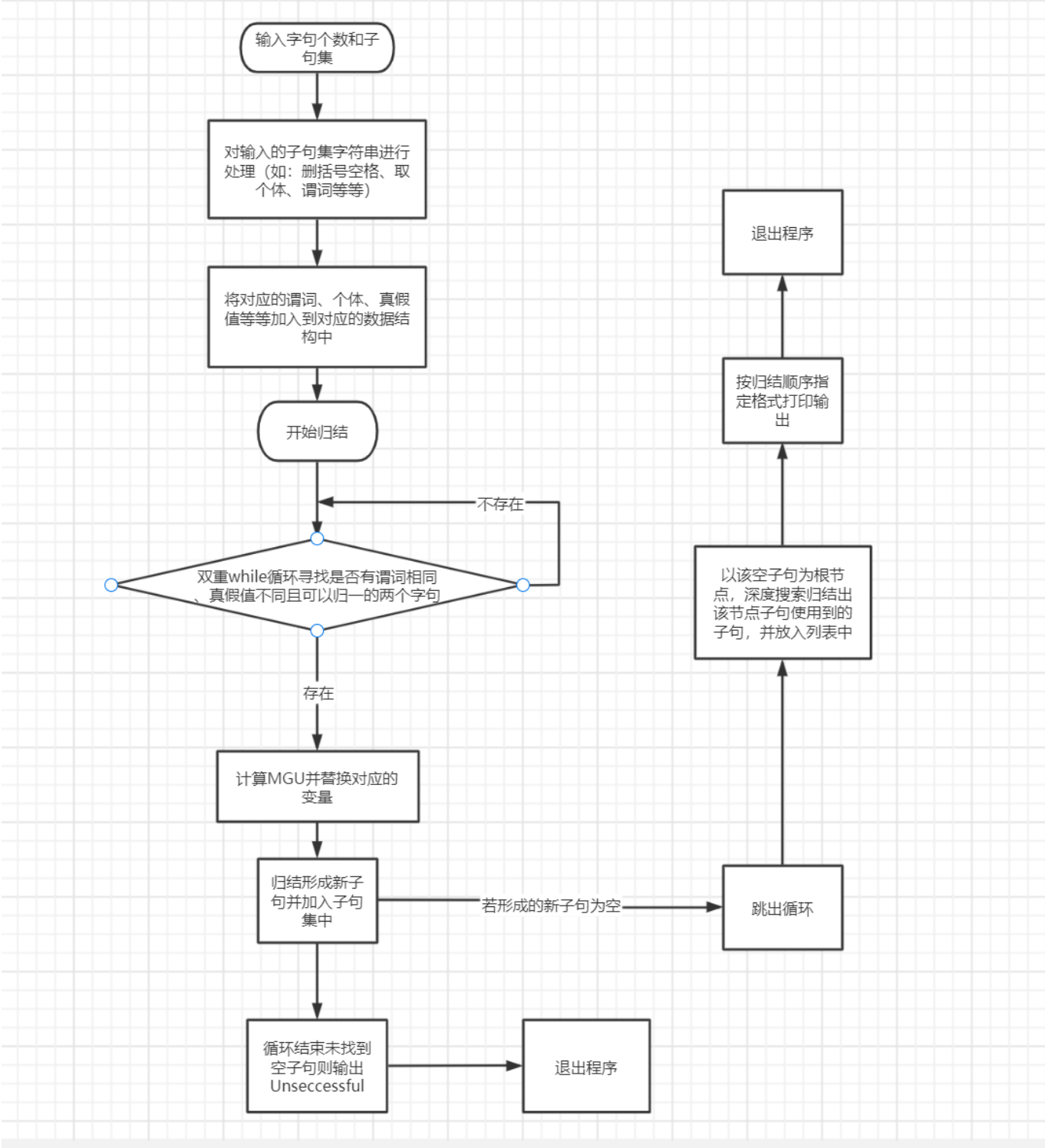
- 命题逻辑归结算法

```
定理：S |-() 当且仅当 S |= (), S |= () 当且仅当 S 是不可满足的
通过该定理，我们可得KB |= α 当且仅当 KB ∧?α 不可满足，于是可以通过反证法证明KB |= α
归结算法：
将α取否定，加入到KB当中
将更新的KB转换为clausal form得到S
反复调用单步归结
如果得到空子句，即S|-()，说明KB ∧?α 不可满足，算法终止，可得KB |= α
如果一直归结直到不产生新的子句，在这个过程中没有得到空子句，则KB |= α不成立
单步归结：
1.使用MGU算法从两个子句中得到相同的原子，及其对应的原子
2.否定去掉该原子并将两个子句合为一个，加入到S子句集合中
3.例如(?Student(x),HardWorker(x))和(HardWorker(sue))合并为(?Student(sue))
```

- 最一般合一算法：

```
合一（unifier）：
通过变量替换使得两个子句能够被归结（有相同的原子），所以合一也被定义为使得两个原子公式等价的一组变量替换/赋值
由于一阶逻辑中存在变量，所以归结之前需要进行合一，如(P(john),Q(fred),R(x))和(?P(y),R(susan),R(y))两个子句中，我们无法找到一样的原子及其对应的否定，但是不代表它们不能够归结
通过将y替换为john，我们得到了(P(john),Q(fred),R(x))和(?P(john),R(susan),R(john))，此时我们两个子句分别存在原子P(john)和它的否定?P(john)，可以进行归结
最一般合一：指使得两个原子公式等价，最简单的一组变量替换
```

2.流程图



3.关键代码展示

1. 创建存放谓词和个体，以及子句的数据结构，以及对应的类函数

```
class Predicate:#放谓词和个体(放项)
    def __new__(cls):
        return super().__new__(cls)
    def __init__(self):
        self.pred = ''#谓词
        self.indiv = []#谓词的个体放入列表
        self.isfunc = []#判断是否是函数
        self.flag= True #判断真假值
```

```
    def printPredicate(self):#返回打印谓词
        tmp = ''
        if(self.flag == False):
            tmp += '~'
        tmp += self.pred
        tmp += '('
        for i in range(len(self.indiv)):
            tmp += self.indiv[i]
            if(not (i == len(self.indiv)-1) ): tmp += ','
        tmp += ')'
        return tmp

class Clause:#放子句
    def __new__(cls):
        return super().__new__(cls)
    def __init__(self) :
        self.index = 0#记录在子句集的下标
        self.pre = []#Predicate的列表
        self.son = []#归结出来的新语句是由哪两个归结的
        self.sonson = []#记录1a，2b中的字母，由归结子句的哪一部分归结
        self.msg = ''#记录归一
        self.printindex = 0#后面输出有用
    def printClause(self):#打印输出子句
        tmp = ''
        if(len(self.pre)>1):tmp += '('
        for i in range(len(self.pre)):
            tmp += self.pre[i].printPredicate()
            if(not(i == len(self.pre)-1 )):tmp += ','
        if(len(self.pre)>1):tmp += ')'
        return tmp
```

2. 几个简单编写的函数

```
def ifvariable(s):#判断是否为变量的简单函数
    flag = False
    variables = ['w','v','u','x','y','z','s']#示例中的变量
    return s in variables

def printlist(lis):#打印输出子句集
    for i in range(len(list)):
        print(i,':',list[i].printClause())
        for j in range(len(list[i].pre)):
            print(lis[i].pre[j].pred,lis[i].pre[j].indiv,lis[i].pre[j].flag)
        print('--------------------------------')

def pri(s1,s2):#判断怎么换个体
    if(not(len(s1) == len(s2) )):
        return len(s1)<len(s2)
    return s1<s2
```

3. 得到空字句后，我们使用深度优先搜索策略，将空子句当作一棵树的根节点,不断对归结出该节点子句的两个子句进行递归，如果搜索的子句也是新子句就加入到列表中

```
def load(ans,index,empty_son):#深度优先探索归结空子句的组成子句

    if(len(ans[index].son) == 0):
        return
    else :
        empty_son.append(copy.deepcopy(ans[index]))
        load(ans,ans[index].son[0],empty_son)
        load(ans,ans[index].son[1],empty_son)
```

4. transform函数将输入的子句集的字符串进行处理，再将对应的信息存储到数据结构中

```
def transform(ans ,f):#将输入读入，并将信息放入对应的类中
    n = int(f.readline())#一行一行读入文件
    print('输入:')
    print(n)
    for j in range(n):
        temp = f.readline()#一行一行读入文件
        print(temp,end='')
        clause= [] #放谓词
        individual = [] #放个体
        temp = temp.strip()#把读入的换行符删掉
        if temp[0] == '(':#删除最外面的括号
            temp = temp[1:len(temp)-1:]

        temp2 = []
        cur = 0
        for i in range(len(temp)-1):#一个字句里如果有多个谓词，将谓词分开放入temp2
            if temp[i] == ')' and temp[i+1] == ',' :
                temp2.append(temp[cur:i+1:])
                cur = i+2
        temp2.append(temp[cur::])

        for a in temp2:#将每个谓词的谓词和个体放入p,e
            temp3 = 0
            for i in range(0,len(a)):
                if a[i] == '(':
                    temp3 = i
                    break
            clause.append(a[0:temp3:])

            temp4 = a[temp3:len(a):]
            if temp4[0] == '(':temp4=temp4[1:len(temp4)-1:]#删掉个体的括号
            temp4 = temp4.split(',')

            for i in range(0,len(temp4)):
                if(temp4[i][0] == ' '):temp4[i] = temp4[i][1::]#删掉空格
            individual.append(temp4)
```

```
        ans.append(Clause())#ans放入子句
        ans[-1].index = len(ans)-1

        for i in range(0,len(clause)):#把谓词和个体放入ans的子句中
            ans[-1].pre.append(Predicate())
            if(clause[i][0] == '~'):
                clause[i] = clause[i][1::]
                ans[-1].pre[-1].flag = False
            elif(len(clause[i]) > 1 and clause[i][1] == '~'):
                clause[i] = clause[i][2::]
                ans[-1].pre[-1].flag = False
            elif(clause[i][0] == ' '):
                clause[i] = clause[i][1::]
            ans[-1].pre[-1].pred = copy.deepcopy(clause[i])
            ans[-1].pre[-1].indiv = copy.deepcopy(individual[len(ans[-1].pre)-1])

        for i1 in range(len(ans[-1].pre)):#判断函数可以先不写
            for i2 in range (len(ans[-1].pre[i1].indiv)):
                    for i3 in range(len(ans[-1].pre[i1].indiv[i2])):
                        if('(' in ans[-1].pre[i1].indiv[i2]
[i3]):ans[-1].pre[i1].isfunc.append(i3)
```

5. 使用内外while循环遍历判断是否可以归结,若可以,则归结产生新子句，空子句时跳出

```
def merge(ans,cur,index):
    j = 0
    size = len(ans)#ans为子句集
    endflag = False#判断是否出现空子句
    while j < len(ans) and not endflag:#归结
        for k in range(len(ans[j].pre)):
            s = ans[j].pre[k].pred
            f = ans[j].pre[k].flag
            size = len(ans)
            i = j+1
            while i < size and not endflag:
                for l in range(len(ans[i].pre)):
                    if(j in ans[i].son):continue
                    s1 = ans[i].pre[l].pred
                    f1 = ans[i].pre[l].flag
                    if(s == s1 and not(f == f1 )):
                        flag = False
                        dic1 = {}#运用字典完成归一操作
                        dic2 = {}
                        for o in range(len(ans[i].pre[l].indiv)):
                            if((ifvariable(ans[i].pre[l].indiv[o]) and not
ifvariable(ans[j].pre[k].indiv[o])) or (ifvariable(ans[j].pre[k].indiv[o]) and not
ifvariable(ans[i].pre[l].indiv[o]))or ans[i].pre[l].indiv[o] ==
ans[j].pre[k].indiv[o] ): #当符合归结条件时，进行归结
                                if(pri(ans[i].pre[l].indiv[o]
,ans[j].pre[k].indiv[o])):
                                    flag = True
```

```
                                    dic1[ans[i].pre[l].indiv[o]] =
ans[j].pre[k].indiv[o]
                                    dic2[ans[j].pre[k].indiv[o]] =
ans[i].pre[l].indiv[o]
                                else:
                                    flag=True
                                    dic1[ans[j].pre[k].indiv[o]] =
ans[i].pre[l].indiv[o]
                                    dic2[ans[i].pre[l].indiv[o]] =
ans[j].pre[k].indiv[o]
                            else :
                                flag = False
                                break

                        if(not flag):
                            continue
                        #形成新子句
                        ans.append(Clause())

                        for p in range(len(ans[j].pre)):
                            if(p == k):continue
                            temp = copy.deepcopy(ans[j].pre[p])
                            for e in range(len(temp.indiv)):
                                if(temp.indiv[e] in dic1):
                                    temp.indiv[e] = dic1[temp.indiv[e]]
                            ans[-1].pre.append(temp)

                        for p in range(len(ans[i].pre)):
                            if(p == l):continue
                            temp = copy.deepcopy(ans[i].pre[p])
                            for e in range(len(temp.indiv)):
                                if(temp.indiv[e] in dic1):
                                    temp.indiv[e] = dic1[temp.indiv[e]]
                            ans[-1].pre.append(copy.deepcopy(temp))

                        for item in dic1.values():
                            if(not(dic2[item] == item)):
                                ans[-1].msg += dic2[item]+ "=" + item + " "
                        p = 0
                        while p < len(ans[-1].pre):
                            p1 = p+1
                            while p1<len(ans[-1].pre):
                                if(ans[-1].pre[p].pred==ans[-1].pre[p1].pred and
ans[-1].pre[p].indiv == ans[-1].pre[p1].indiv):
                                    del ans[-1].pre[p1]
                                    p1 -= 1
                                p1 += 1
                            p += 1
                        ans[-1].son.append(j)
                        ans[-1].son.append(i)
                        ans[-1].sonson.append(k)
                        ans[-1].sonson.append(l)
                        ans[-1].index=len(ans)-1
                        if(ans[-1].printClause()==''):#若为空子句跳出
```

```
                            endflag = True
                            index = len(ans)-1
                i = i+1
        j = j+1
    return index
```

6. 深度优先搜索出归结出空子句所需要的子句，再将其按先后顺序排序，最后按指定格式输出结果

```
def printans(cur,ans,index,alpha):
    cur1 = cur
    b= []
    load(ans,index,b)#深度优先搜索出归结出空子句所需要的子句
    b.sort(key=lambda x:(x.son[0],x.son[1]))#按归结出该子句的子句的先后顺序排序
    for i in range(len(b)):
        print('R[',end='')
        if(b[i].son[0] >= cur):
            flag=False
            for j in range(i):
                if(ans[b[i].son[0]].index == b[j].index):
                    print(b[j].printindex,end='')
                    flag = True
                    break
        else :
            print(b[i].son[0]+1,end='')
        print(alpha[b[i].sonson[0]],end='')
        print(',',end='')

        if(b[i].son[1] >= cur):
            flag=False
            for j in range(i):
                if(ans[b[i].son[1]].index == b[j].index):
                    print(b[j].printindex,end='')
                    flag=True
                    break
        else :
            print(b[i].son[1]+1,end='')

        print(alpha[b[i].sonson[1]],end='')
        print(']',end='')
        print('{',end='')
        print(b[i].msg,end='')
        print('}=',end='')
        print(b[i].printClause())
        cur1 += 1
        b[i].printindex = cur1
```

7. 主函数

```
#主函数
def main():
    alpha = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
    f = open(r"ai\E3\code\input.txt",'r')
    ans = []#存放子句集
    transform(ans,f)#将输入读入且转化
    cur = len(ans)#记录初始字句个数
    index = -1#记录空子句子下标
    index = merge(ans,cur,index)#归结
    if index == -1:
        print("Unsuccessful")#无法归结出空子句
    else:
        print('输出:')
        printans(cur,ans,index,alpha)#输出答案
```

4.创新点&优化（如果有）

1. 使用了深度优先搜索的方法，避免了输出了多余的结果，得到空字句后，我们使用深度优先搜索策略，将空子句当作一棵树的根节点,不断对归结出该节点子句的两个子句进行递归，如果搜索的子句也是新子句就加入到列表中,就像是对一棵以空子句为根节点的树的深度遍历

```
def load(ans,index,empty_son):#深度优先探索归结空子句的组成子句

    if(len(ans[index].son) == 0):
        return
    else :
        empty_son.append(copy.deepcopy(ans[index]))
        load(ans,ans[index].son[0],empty_son)
        load(ans,ans[index].son[1],empty_son)
```

2. 对简单使用函数的情况进行了考虑，但对复杂一些的如函数嵌套还未解决

三、实验结果及分析

1. 实验结果展示示例

1.
    - 试验任务一：Alpine Club

```
PS C:\Users\刘俊杰\Desktop\code> C:\Users\刘俊杰\AppData\Local\Programs\Python\Python39\python.exe "c:\Users\刘俊杰\Desktop\code\ai\E3\e3.py"
输入：
11
A(tony)
A(mike)
A(john)
L(tony, rain)
L(tony, snow)
(~A(x), S(x), C(x))
(~C(y), ~L(y, rain))
(L(z, snow), ~S(z))
(~L(tony, u), ~L(mike, u))
(L(tony, v), L(mike, v))
(~A(w), ~C(w), S(w))
输出：
R[2a,6a]{x=mike }=(S(mike),C(mike))
R[2a,11a]{w=mike }=(~C(mike),S(mike))
R[5a,9a]{u=snow }=~L(mike,snow)
R[8a,14a]{z=mike }=~S(mike)
R[12b,13a]{}=S(mike)
R[15a,16a]{}=
```

2. 试验任务二：Graduate Student

```
PS C:\Users\刘俊杰\Desktop\code> C:\Users\刘俊杰\AppData\Local\Programs\Python\Python39\python.exe "c:\Users\刘俊杰\Desktop\code\ai\E3\e3.py"
输入：
4
GradStudent(sue)
(~GradStudent(x), Student(x))
(~Student(x), HardWorker(x))
~HardWorker(sue)
输出：
R[1a,2a]{x=sue }=Student(sue)
R[3a,5a]{x=sue }=HardWorker(sue)
R[4a,6a]{}=
```

3. 试验任务三：Block World

```
PS C:\Users\刘俊杰\Desktop\code> C:\Users\刘俊杰\AppData\Local\Programs\Python\Python39\python.exe "c:\Users\刘俊杰\Desktop\code\ai\E3\e3.py"
输入：
5
On(aa,bb)
On(bb,cc)
Green(aa)
~Green(cc)
(~On(x,y), ~Green(x), Green(y))
输出：
R[1a,5a]{x=aa y=bb }=(~Green(aa),Green(bb))
R[2a,5a]{x=bb y=cc }=(~Green(bb),Green(cc))
R[3a,6a]{}=Green(bb)
R[4a,7b]{}=~Green(bb)
R[8a,9a]{}=
```

2. 评测指标展示及分析

1. 未优化前： 时间复杂度：这程序时间的耗费主要在归结部分，在归结部分使用了内外循环，设归结后子句的数量为n,子句中最大的谓词个数为m,谓词中最大的个体数量为l，则时间复杂度最坏情况下可以达到O(n^2*m*^2l^2)，时间复杂度高。

- 空间复杂度：空间主要是在存储子句的数据结构的空间

2. 优化后：

- 上方实验结果展示示例便是优化后的
- 优化后的对时间和空间复杂度没什么影响，主要是将需要的输出了，不需要的输出了
- 分析：对没有使用函数的复杂子句集的归结来说可以输出正确的结果，可以输出需要的子句归结步骤，但理论上来说时间复杂度高，对于助教和老师提到的使用启发式搜索没有什么思路，如果要这个程序进行进一步的改善，就要使用更加高效的搜索算法

四、思考题

1. 学有余力1

```
I(bb)
U(aa,bb)
?F(u)
(?I(y),?U(x,y), F(f(z)))
(?I(v), ?U(w,v),E(w,f(w)))
R[3,4c]{u=f(z)} (?I(y),?U(x,y))
R[1,6b]{y=bb} ?U(x,bb)
R[2,7] {x=aa} [ ]
```

运行情况：

```
PS C:\Users\刘俊杰\Desktop\code> C:\Users\刘俊杰\AppData\Local\Programs\Python\Python39\python.exe "c:\Users\刘俊杰\Desktop\co
de\ai\E3\e3.py"
输入：
5
I(bb)
U(aa,bb)
~F(u)
(~I(y),~U(x,y), F(f(z)))
(~I(v), ~U(w,v),E(w,f(w)))
输出：
R[1a,4a]{y=bb }=(~U(x,bb),F(f(z)))
R[2a,6a]{x=aa }=F(f(z))
R[3a,7a]{u=f(z) }=
```

2. 学有余力2

```
?P(aa)
(P(z), ?Q(f(z),f(u)))
(Q(x, f(g(y))), R(s))
?R(t)
R[1,2a]{z=aa} ?Q(f(aa), f(u))
R[3,5]{x=f(aa)} (Q(f(aa), f(g(y))), R(s))
R[5,6a] {u=g(y)} R(s)
R[4,7] {s=t} []
```

- 针对函数嵌套的问题还未解决

五、参考资料

- 理论课归结和归一部分的课件
- 实验课课件