20337270_钟海财_实验2

中山大学计算机学院

本科生实验报告

(2021学年春季学期)

课程名称：Artificial Intelligence

教学班级	**20级软工+网安**	**专业（方向）**	**软件工程**
学号	**20337270**	**姓名**	**钟海财**

一、 实验题目

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}.

实验任务1 Aipine Club

On(aa,bb)
On(bb,cc)
Green(aa)
¬Green(cc)
(¬On(x,y),¬Green(x),Green(y))

实验任务2 Graduate Student

GradStudent(sue)
(¬GradStudent(x), Student(x))
(¬Student(x),HardWorker(x))
¬HardWorker(sue)

实验任务3 Block World

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))

注：在本次实验中，满足len(s1) = 1 的 s1表示自由变量，满足len(s2) > 1 的s2表示约束变量

二、 实验内容

1. 算法原理

□ 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

□ Clausal form (便于计算机处理的形式)

□ 每一个子句对应一个元组，元组每一个元素是一个原子公式/原子公式的否定， 元素之间的关系是析取关系，表示只要一个原子成立，该子句成立 □ 如子句¬child∨¬male∨boy对应数据结构(¬child,¬male,boy)，空子句()对应False

□ 元组的集合组成子句集S，子句集中每个句子之间是合取关系，表示每一个 子句都应该被满足

□ 由于本次实验重点是归结算法，所以问题输入是已经转换过的clausal form， 关于具体转换方式感兴趣的同学可以参考课件

□ 单步归结

□ 从两个子句中分别寻找相同的原子，及其对应的原子否定

□ 去掉该原子并将两个子句合为一个，加入到S子句集合中

□ 例如(¬child,¬female,girl)和(child)合并为(¬female,girl)

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

归结原则公式：

对于子句集：

$$((P, C_1), (\neg P, C_2), \dots)$$

归结原则：

$$(P, C_1) \wedge (\neg P, C_2) \Rightarrow (C_1, C_2)$$

归结原则的一种特殊形式：

$$(P, C_1) \wedge \neg P \Rightarrow C_1$$

2. 伪代码/流程图

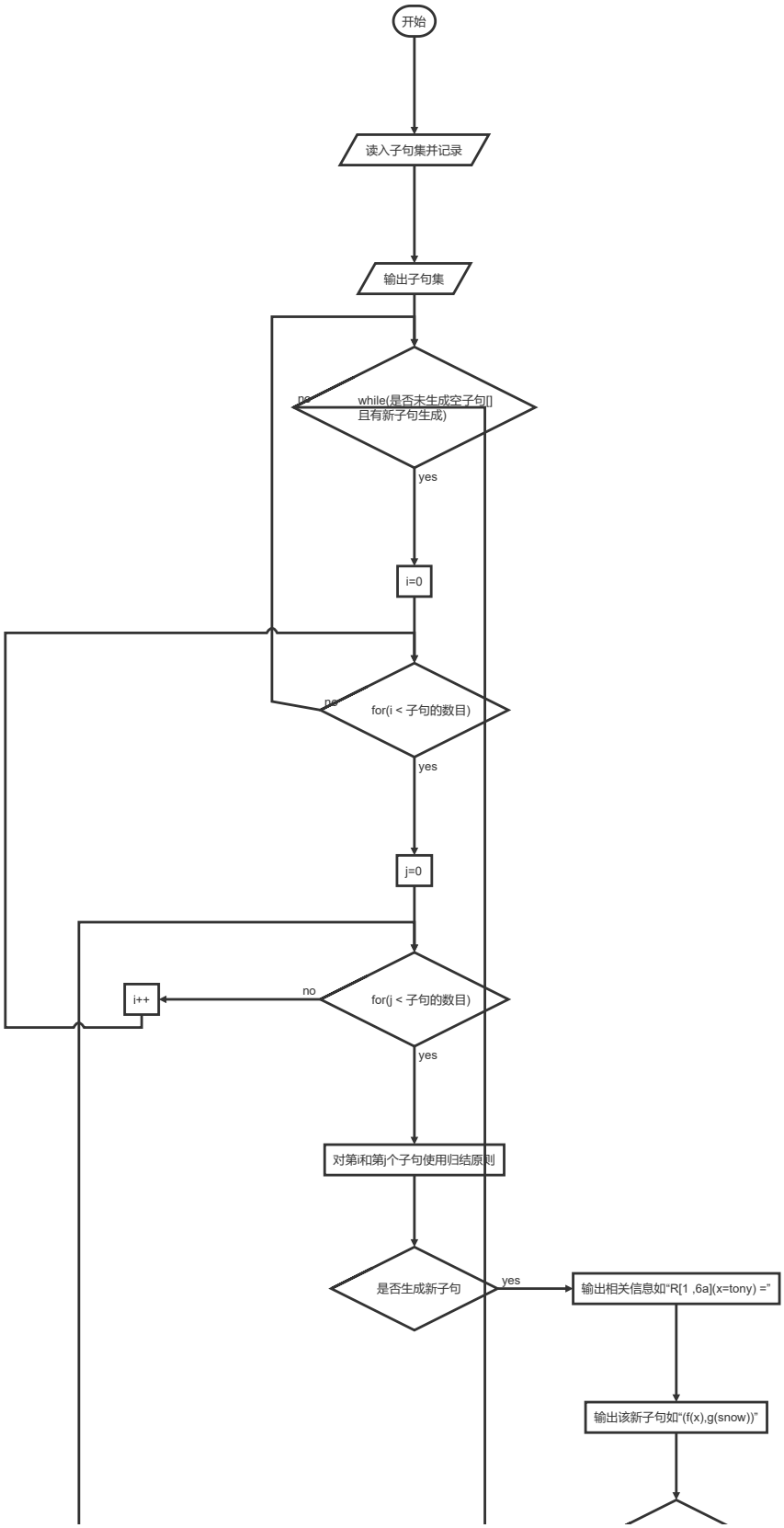
伪代码（语法可能有些错误）：

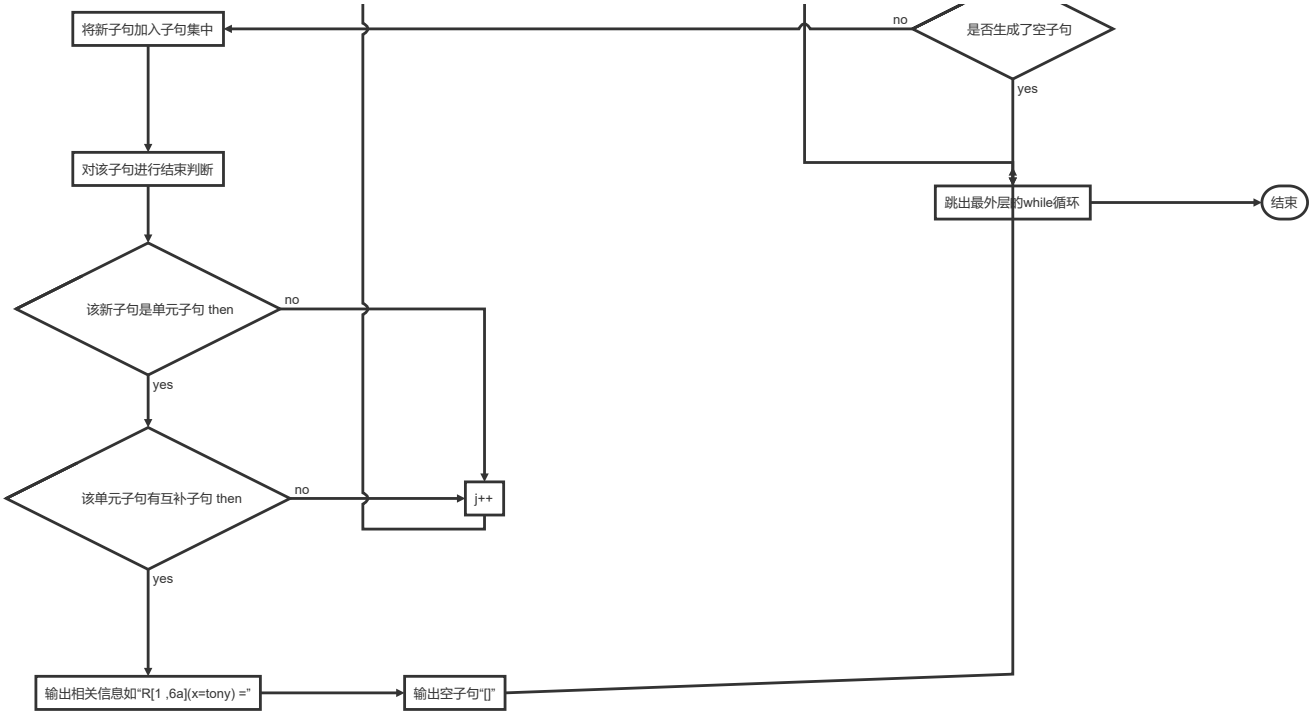
```
begin
clause_set = [] // 用于记录子句集
while 有子句输入
    do 读入子句并记录到子句集中clause_set
    end
for var i=0 to 子句的数目 // 输出子句集，判断前面对子句集的记录是否正确
```

```
    do 输出第i个子句
    end

while 未生成空子句且有新子句生成
    for i to 子句的数目
        for j to 子句的数目 // 对第i和第j个子句使用归结原则
            do 对第i和第j个子句使用归结原则
            if 生成新子句 then
                do 输出相关信息如“R[1 ,6a](x=tony) =”
                do 输出该新子句如“(f(x),g(snow))”
            if 生成了空子句 then
                do 跳出最外层的while循环
                do 将新子句加入子句集中
                do 对该子句进行结束判断
            if 该新子句是单元子句 then
                if 该单元子句有互补子句 then
                    do 输出相关信息如“R[1 ,6a](x=tony) =”
                    do 输出空子句“[]”
                    do 跳出最外层的while循环
            end
        end
    if (没有新子句生成) then
        do 跳出外层while循环
        输出"Not True"
    end

end
```

##

流程图


开始
读入子句集并记录
输出子句集
while(是否未生成空子句[]
且有新子句生成)
no
yes
i=0
for(i < 子句的数目)
no
yes
j=0
for(j < 子句的数目)
no
i++
yes
对第i和第j个子句使用归结原则
是否生成新子句
yes
输出相关信息如"R[1 ,6a](x=tony) ="
输出该新子句如"(f(x),g(snow))"


将新子句加入子句集中
对该子句进行结束判断
该新子句是单元子句 then
no
yes
该单元子句有互补子句 then
no
j++
yes
输出相关信息如“R[1 ,6a](x=tony) =”
输出空子句“[]”
no
是否生成了空子句
yes
跳出最外层的while循环
结束

##

3. 关键代码展示（带注释）

```
class Clause:  # 单个谓词的类 如 ~L(a,bb,y) -> ['~L','a','bb','y']
    element = []  # element[0]谓词名(含~号)如~L, element[1:]变量名 如 a , bb , y

    def __init__(self, str1):   # 用字符串进行构造
        self.element = []
        if str1[0] == ',':
            str1 = str1[1:]
        s0 = ""
        for i in range(len(str1)):
            s0 += str1[i]
            if str1[i] == '(' or str1[i] == ',' or str1[i] == ')':
                self.element.append(s0[0:-1])
                s0 = ""

    def create(self, str1):     # 用字符串进行赋值
        self.element = []
        if str1[0] == ',':
            str1 = str1[1:]
        s0 = ""
        for i in range(len(str1)):
            s0 += str1[i]
            if str1[i] == '(' or str1[i] == ',' or str1[i] == ')':
                self.element.append(s0[0:-1])
                s0 = ""

    def create2(self, list2):   # 用列表进行赋值
        self.element = []
        for i in range(len(list2)):
            self.element.append(list2[i])

    def change_name(self, old_name0, new_name0):    # 自由变量换名，进行合一置换
        for i in range(len(old_name0)):
            j = 1
            while j < len(self.element):
                if self.element[j] == old_name0[i]:
                    self.element[j] = new_name0[i]
                j = j + 1

    def pre(self):  # 返回谓词的前缀是否为"¬"
        return self.element[0][0] == "¬"

    def getname(self):  # 返回谓词名称
        if self.pre():
            return self.element[0][1:]
        else:
            return self.element[0]

    def get_pre_name(self):     # 返回全称含"¬"
        return self.element[0]
```

```
def out_clause(new_clause):      # 输出一个子句
    if len(new_clause) > 1:
        print("(", end="")
    for j1 in range(len(new_clause)):  # 输出该新子句
        print(new_clause[j1].element[0], end="(", sep="")
        for j2 in range(1, len(new_clause[j1].element)):
            print(new_clause[j1].element[j2], end="")
            if j2 < len(new_clause[j1].element) - 1:
                print(",", end="")
        print(")", end="")
        if j1 < len(new_clause) - 1:
            print(",", end="")
    if len(new_clause) > 1:
        print(")", end="")
    if len(new_clause) > 0:
        print("")


def out_msg(key1, key2, i, j, old_name, new_name, clause_set):    # 输出合一置换的相关信息如 R[1 ,6a](x=tony) =
    print(len(clause_set), end=":")
    c1 = chr(key1 + 97)   # 将key1,key2(发生置换的谓词在子句中的位置)转化为字符
    c2 = chr(key2 + 97)
    print("\tR[", i + 1, end="", sep="")
    if len(clause_set[i]) != 1:     # len(new_name) == 0 and
        print(c1, end="", sep="")
    print(", ", j + 1, end="", sep="")
    if len(clause_set[j]) != 1:
        print(c2, end="", sep="")
    print("]", end="(", sep="")
    for i2 in range(len(old_name)):
        print(old_name[i2], "=", new_name[i2], end="", sep="")
        if i2 < len(old_name)-1:
            print(", ", end="")
    print(") = ", end="", sep="")


# end1:每生成一个单谓词子句（单元子句）,若能找到对应的互补子句(不考虑合一置换,只考虑完全从字面上互补的子句,合一置换由下一次while循环考虑),则生成空子句[]结束
# end2:在用归结原理（单元优先策略）进行归结时直接生成了一个空子句[]则结束(当且仅当子句集中就已经存在两个互补的单元子句。含合一置换后互补的)
def is_break(new_clause, clause_set):   # 判读是否结束end1/end2,
    if len(new_clause) == 0:  # 结束判断end2
        print("  []\nend2")
        return True
    if len(new_clause) == 1:  # 进行结束判断：若有两句互斥的单谓词子句，则产生空子句
        for j2 in range(len(clause_set) - 1):  # clause_set[j]超过一个谓词的取或的子句
            if len(clause_set[j2]) == 1 and new_clause[0].getname() == clause_set[j2][0].getname() \
                    and new_clause[0].element[1:] == clause_set[j2][0].element[1:] \
                    and new_clause[0].pre() != clause_set[j2][0].pre():
                print(len(clause_set) + 1, ":\tR[", j2 + 1, ", ", len(clause_set), "]() = []\nend1", sep="")
                # keep = False
                return True
    return False


def exit_yet(new_clause, clause_set, key2):  # 此处判断是否生成了一个已经存在的子句
    key = key2
    if len(new_clause) == 1:
```

```
        for j in range(len(clause_set)):
            if len(clause_set[j]) == 1 and new_clause[0].element == clause_set[j]
[0].element:
                key = -1
                break
    return key


# 对clause1和clause2进行合一置换，单步归结生成新子句new_clause
def mgu(clause1, clause2, key1, key2, old_name, new_name, new_clause):  #
(¬A,C1)∧(A,C2) => (C1,C2)
    num = 0
    key1.clear()    # 记录置换的变量在子句clause1中的位置
    key2.clear()    # 记录置换的变量在子句clause2中的位置
    old_name.clear()    # 记录发生置换的自由变量名称
    new_name.clear()    # 记录发生置换的约束变量名称
    for i in range(len(clause1)):   # 找两子句中互补的两个谓词
        for j in range(len(clause2)):
            if clause1[i].pre() != clause2[j].pre() and clause1[i].getname() ==
clause2[j].getname():
                old_name.append([]) # 自由变量名称
                new_name.append([]) # 约束变量名称
                num += 1    # 记录互补的谓词的对数
                key1.append(i)  # 记录变量的位置
                key2.append(j)  # 记录变量的位置
                for k in range(len(clause1[i].element)-1):
                    t = len(old_name)-1
                    if clause1[i].element[k + 1] == clause2[j].element[k + 1]:
                        continue
                    if len(clause1[i].element[k + 1]) == 1:
                        old_name[t].append(clause1[i].element[k + 1])
                        new_name[t].append(clause2[j].element[k + 1])
                    elif len(clause2[j].element[k + 1]) == 1:
                        old_name[t].append(clause2[j].element[k + 1])
                        new_name[t].append(clause1[i].element[k + 1])
                    elif clause1[i].element[k + 1] != clause2[j].element[k + 1]:
                        num = num-1
                        key1[t] = -1
                        key2[t] = -1
                        break
                break
    new_clause.clear()
    the_key = -1
    if num == 1:    # num>1时说明生成的新子句中含有互补的两个谓词，真值恒为1；num=0表示没
有互补的两个谓词
        for i in range(len(key1)):  # 生成一个新子句
            if key1[i] != -1:
                the_key = i
                for k1 in range(len(clause1)):
                    if k1 != key1[i]:
                        c1 = Clause("sb")
                        c1.create2(clause1[k1].element)
                        c1.change_name(old_name[i], new_name[i])
                        new_clause.append(c1)
                for k2 in range(len(clause2)):
                    if k2 != key2[i]:
                        c2 = Clause("sb")
                        c2.create2(clause2[k2].element)
                        c2.change_name(old_name[i], new_name[i])
                        yet = False
                        for j in range(len(new_clause)):    # 去除重复的谓词
```

```
                            if new_clause[j].element == c2.element:
                                yet = True
                        if not yet:
                            new_clause.append(c2)
                break
    return the_key


# ¬
def main():
    clause_set = []     # 储存子句集,clause_set[i]表示子句集中的第i个子句
    print("Please enter a clause set")
    clause0 = input()  # 读入一个子句 , 如(A(x),B(a,z),PPT(s,b)) 或 A(x)
    i = 0
    while clause0:  # 读入子句集并记录
        clause_set.append([])  # 加入一个空列表用于记录第i个子句，该列表的元素类型为谓词公式类Clause
        if clause0[0] == '(':  # 去掉最外边的括号(如果有的话)
            clause0 = clause0[1:-1]
        str0 = ""  # 用于记录一个谓词公式,如 ",~B(a,z)" 最开头的“,”在create函数里消去
        for j in range(len(clause0)):
            if clause0[j] == " ":  # 跳过空格
                continue
            str0 += clause0[j]
            if clause0[j] == ')':  # 用')'作为结尾分割成多个谓词公式
                clause1 = Clause(str0)  # 创造一个谓词公式类Clause的变量
                clause_set[i].append(clause1)  # 加入到子句集的第i个子句中
                str0 = ""
        i = i + 1  # i == len(clause_set)-1
        clause0 = input()

    for i in range(len(clause_set)):  # 输出子句集，判断前面对子句集的记录是否正确
        out_clause(clause_set[i])

    last_end = []  # 用于记录上一次while循环的第一重for循环每次结束的子句集的大小，作为下次while循环单位词子句第二重for循环开始的位置
    for i in range(len(clause_set)):
        last_end.append(0)
    keep = True
    model = 0   # 用于转换归并原理的策略，model==0时使用单元优先策略，model==1时使用普遍策略
    while keep:
        last_len = len(clause_set)  # 记录循环开始的子句集的大小
        # 采用单元优先策略,先只用单元子句和其他子句进行归结，没有新子句产生时，再使用一般的归结原则
        for i in range(len(clause_set)):
            if model == 0 and len(clause_set[i]) != 1:  # 单元优先策略跳过多谓词子句
                continue
            if model == 1 and len(clause_set[i]) == 1:  # 普遍形式跳过单元子句
                continue
            if not keep:
                break
            this_start = last_end[i]  # 本次while循环的第i子句的开始是上次while循环第i子句的结束
            last_end[i] = len(clause_set)  # 本次的第i子句的结束是下次第i子句的开始
            for j in range(this_start, len(clause_set)):  # clause_set[j]超过一个谓词的取或的子句
                if not keep:
                    break
                if model == 0 and i == j:       # 单元优先策略
                    continue
```

```
                if model == 1 and (i >= j or len(clause_set[j])) == 1:  # 普遍形式
                    continue
                key1, key2, old_name, new_name, new_clause = [], [], [], [], []
                the_key = mgu(clause_set[i], clause_set[j], key1, key2, old_name,
new_name, new_clause)
                the_key = exit_yet(new_clause, clause_set, the_key)
                if the_key == -1:
                    continue
                clause_set.append(new_clause)
                last_end.append(0)
                if len(new_clause) == 0 and len(clause_set[i]) == 1 and
len(clause_set[j]) == 1:
                    out_msg(key1[the_key], key2[the_key], i, j, old_name[the_key],
new_name[the_key], clause_set)
                    # 输出置换的相关信息如 R[3, 6a] =
                    out_clause(new_clause)  # 输出该新子句
                elif len(new_clause) == 0:  # 表示没有生成新子句
                    continue
                else:
                    out_msg(key1[the_key], key2[the_key], i, j, old_name[the_key],
new_name[the_key], clause_set)
                    # 输出置换的相关信息如 R[3, 6a] =
                    out_clause(new_clause)  # 输出该新子句
                if is_break(new_clause, clause_set):    # 进行结束判断
                    keep = False
                    break
        if model == 0 and last_len == len(clause_set):     # 如果没有新子句产生，再使
用普遍形式
            model = 1
        elif model == 1 and last_len == len(clause_set):     # 仍然没有新的子句产生
时，说明无法得到[]，说明无法证明子句集真值为假
            keep = False
            print("Not True")
        else:   # 否则，采取单元优先策略
            model = 0
    print("OK!")


if __name__ == '__main__':
    main()
```

4. 创新点&优化（如果有）

1.采用单元优先策略：

（1）首先只用单元子句和其他子句进行归结，

（2）若没有新子句产生，再对非单元子句间进行归结。

重复（1）、（2）直至生成空子句或没有新子句生成。

由于本次实验的3个测试样例仅用(1)就能归结出空子句，所以采用单元优先策略能更快得到答案（当能生成空子句时）。

2.循环结束判断：

一般来说是在归结过程中归结出一个空子句[]作为循环结束的，但是每一个空子句都是由两个互补的单元子句进行归结产生，所以更改循环结束判断策略：

如果生成的子句是单元子句，对已存在的单元子句进行遍历，如果找到与之互补的单元子句，说明这两个子句可归结生成一个空子句，循环结束（end1）；

否则如果生成的子句是空子句，且是由两个单元子句归结生成的，循环结束（end2）；

否则循环继续。

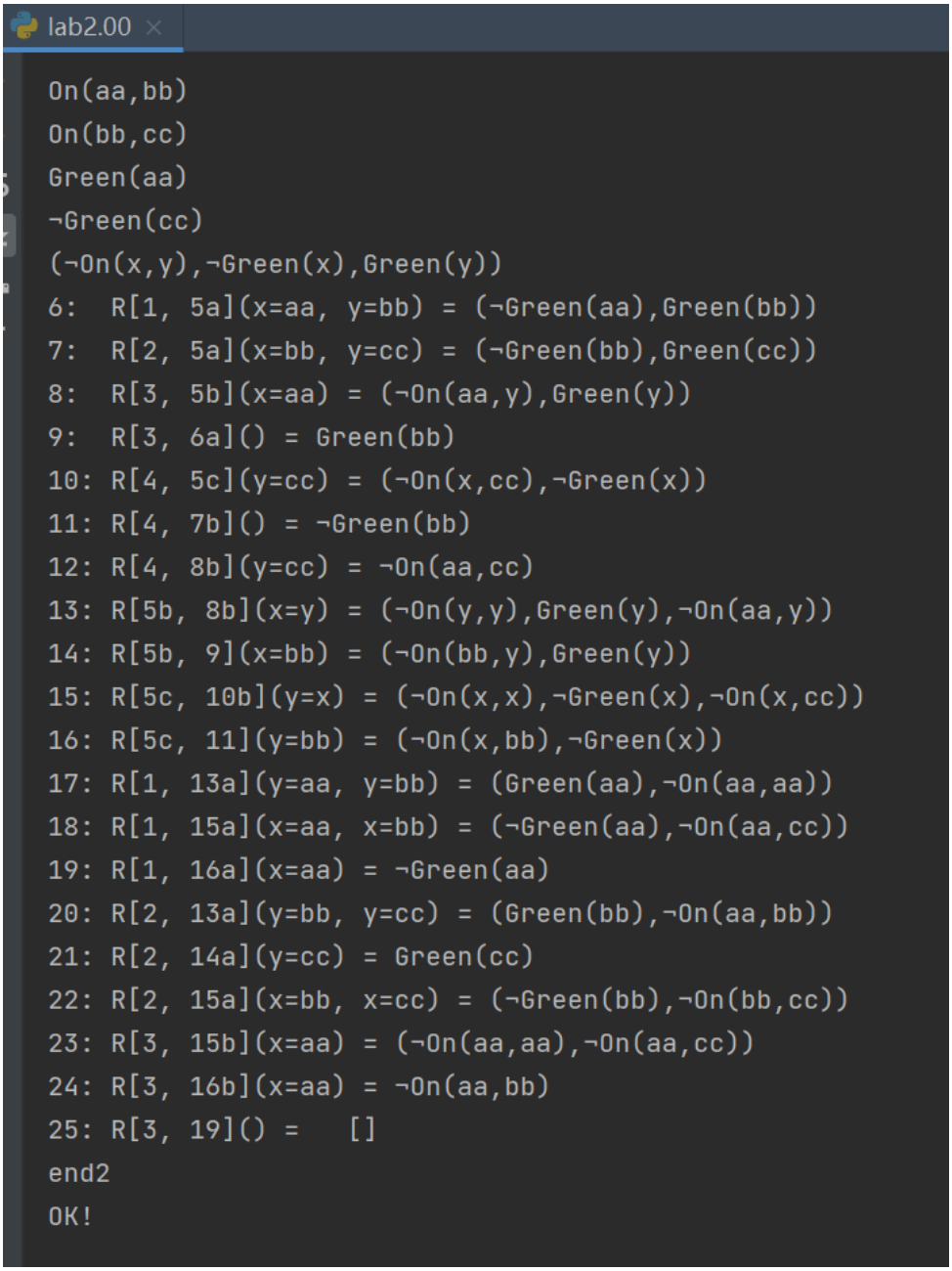
一般以end1结束能比以end2结束少一轮while循环。

三、 实验结果及分析

1.无优化

1. 实验结果展示示例（可图可表可文字，尽量可视化）

实验任务1 Aipine Club



实验任务2 Graduate Student

```
lab2.00
D:\Anaconda\python.exe D:/桌面文件/python_project/lab2.00.py
Please enter a clause set
GradStudent(sue)
(¬GradStudent(x), Student(x))
(¬Student(x),HardWorker(x))
¬HardWorker(sue)

GradStudent(sue)
(¬GradStudent(x),Student(x))
(¬Student(x),HardWorker(x))
¬HardWorker(sue)
5:  R[1, 2a](x=sue) = Student(sue)
6:  R[2b, 3a]() = (¬GradStudent(x),HardWorker(x))
7:  R[3b, 4](x=sue) = ¬Student(sue)
8:  R[3a, 5](x=sue) = HardWorker(sue)
9:  R[4, 6b](x=sue) = ¬GradStudent(sue)
10: R[4, 8]() =   []
end2
OK!
```

实验任务3 Block World

```
lab2.00
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))
12: R[1, 6a](x=tony) = (S(tony),C(tony))
13: R[1, 11a](w=tony) = (¬C(tony),S(tony))
14: R[2, 6a](x=mike) = (S(mike),C(mike))
15: R[2, 11a](w=mike) = (¬C(mike),S(mike))
16: R[3, 6a](x=john) = (S(john),C(john))
17: R[3, 11a](w=john) = (¬C(john),S(john))
18: R[4, 7b](y=tony) = ¬C(tony)
19: R[4, 9a](u=rain) = ¬L(mike,rain)
20: R[5, 9a](u=snow) = ¬L(mike,snow)
```

```
111:    R[2, 58b]() = ¬L(tony,snow)
112:    R[2, 81a](z=mike) = (S(mike),L(mike,snow))
113:    R[2, 84a]() = (S(mike),L(mike,snow))
114:    R[2, 86a]() = ¬C(mike)
115:    R[2, 88a]() = (S(mike),¬L(tony,snow))
116:    R[2, 91a]() = (S(mike),L(mike,snow))
117:    R[3, 21a](y=john) = (S(john),¬L(john,rain))
118:    R[3, 22a](z=john) = (C(john),L(john,snow))
119:    R[3, 23a](w=john) = S(john)
120:    R[3, 32b](z=john) = (¬L(john,rain),L(john,snow))
121:    R[3, 34b](w=john) = (L(john,snow),¬C(john))
122:    R[3, 42b](y=john) = (L(john,snow),¬L(john,rain))
123:    R[3, 43b](w=john) = L(john,snow)
124:    R[3, 81a](z=john) = (S(john),L(john,snow))
125:    R[3, 85a]() = (S(john),L(john,snow))
126:    R[4, 21c](y=tony) = (¬A(tony),S(tony))
127:    R[4, 32a](z=tony) = (¬A(tony),L(tony,snow))
128:    R[4, 42c](y=tony) = (L(tony,snow),¬A(tony))
129:    R[5, 55a]() = C(mike)
130:    R[5, 58a]() = ¬A(mike)
131:    R[5, 88c]() = (¬A(mike),S(mike))
132:    R[5, 111]() =   []
end2
OK!
```

2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

评测指标：

最后两个互补的单元子句产生的位置及最终空子句产生的位置，三项数据可由最后空集产生所在行得知，如：

28: R[26,27]（）= [] 得到指标（26，27，28）

3个任务的评测指标分别为：（3，19，25）（4，8，10）（5，111，132）

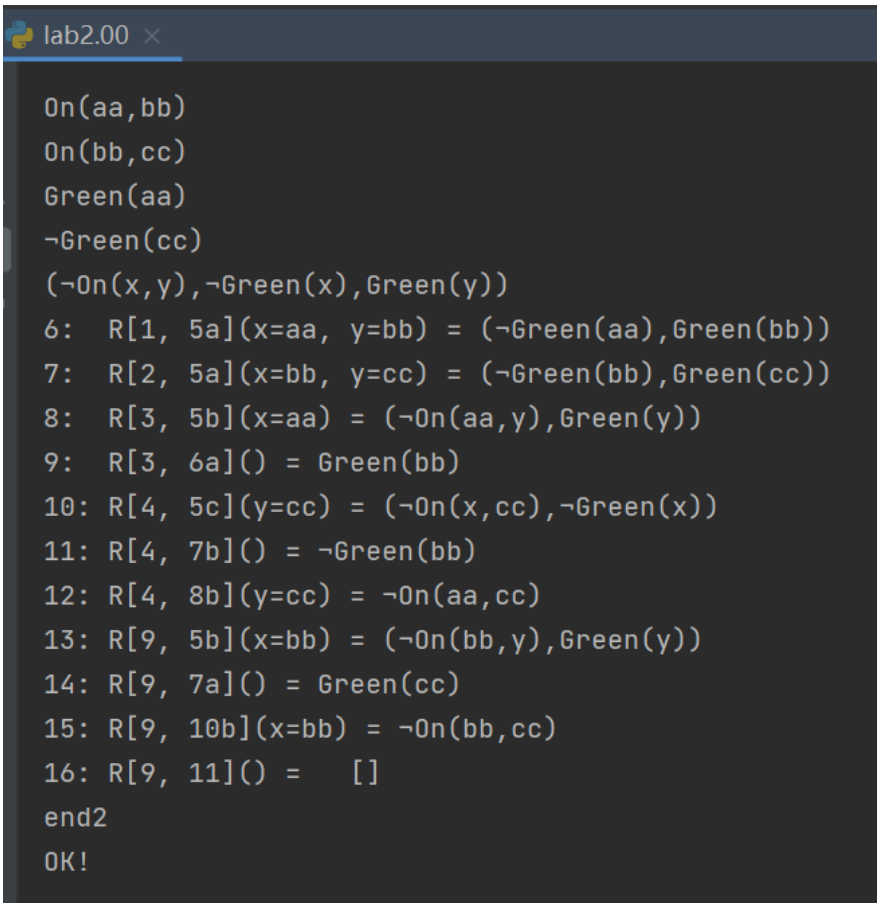
分析：

在任务1，3中，第二个互补单元子句产生的位置明显与第一个互补单元子句的位置相差较远，同时最后空集产生的位置与第二个互补单元子句产生的位置也明显相差较远，说明算法在产生互补单元子句的策略以及判断结束的策略不是很好。

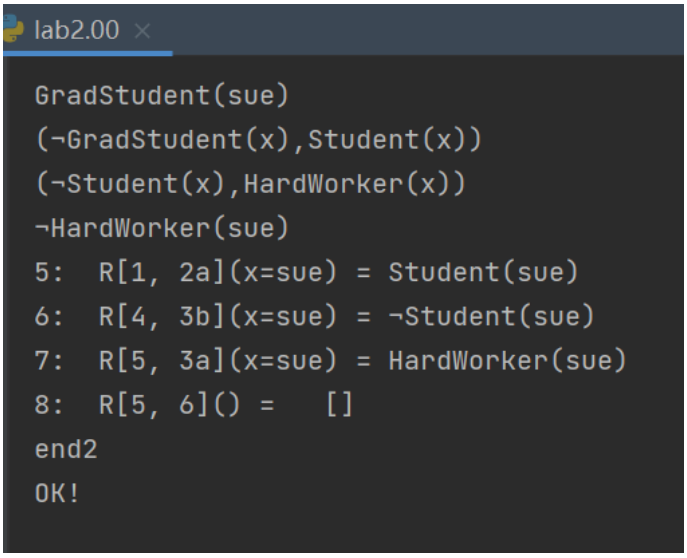
2.仅用“1.采用单元优先策略 ”优化

1. 实验结果展示示例（可图可表可文字，尽量可视化）

实验任务1 Aipine Club



实验任务2 Graduate Student



实验任务3 Block World

lab2.00

```
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))
12: R[1, 6a](x=tony) = (S(tony),C(tony))
13: R[1, 11a](w=tony) = (¬C(tony),S(tony))
14: R[2, 6a](x=mike) = (S(mike),C(mike))
15: R[2, 11a](w=mike) = (¬C(mike),S(mike))
16: R[3, 6a](x=john) = (S(john),C(john))
17: R[3, 11a](w=john) = (¬C(john),S(john))
18: R[4, 7b](y=tony) = ¬C(tony)
19: R[4, 9a](u=rain) = ¬L(mike,rain)
20: R[5, 9a](u=snow) = ¬L(mike,snow)
21: R[18, 6c](x=tony) = (¬A(tony),S(tony))
22: R[18, 12b]() = S(tony)
23: R[20, 8a](z=mike) = ¬S(mike)
24: R[23, 6b](x=mike) = (¬A(mike),C(mike))
25: R[23, 11c](w=mike) = (¬A(mike),¬C(mike))
26: R[23, 14a]() = C(mike)
27: R[23, 15b]() = ¬C(mike)
28: R[26, 11b](w=mike) = (¬A(mike),S(mike))
29: R[26, 15a]() = S(mike)
30: R[26, 25b]() = ¬A(mike)
31: R[26, 27]() =   []
end2
OK!
```

2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

评测指标：

最后两个互补的单元子句产生的位置及最终空子句产生的位置，三项数据可由最后空集产生所在行得知，如：

28: R[26,27]（）= [] 得到指标（26，27，28）

3个任务的评测指标分别为：（9，11，16）（5，6，8）（26，27，31）

分析：

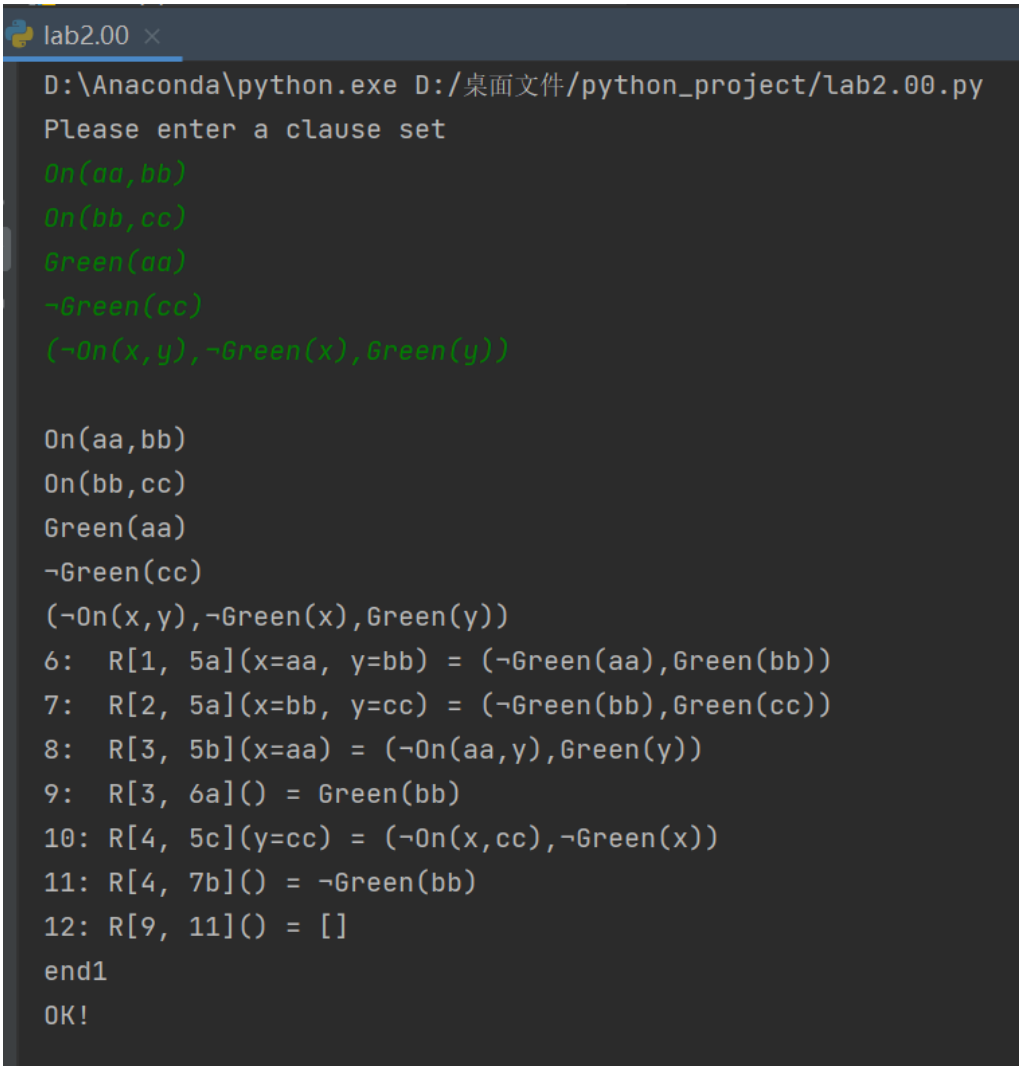
将评测指标与无优化的评测指标进行对比：（3，19，25）（4，8，10）（5，111，132）

显然3个任务最后产生空子句的位置都得到极大提前，同时3个位置两两之间的距离都得到缩短，说明本次实验中使用“1.采用单元优先策略 ”优化对于互补单元子句的产生有着较好的效果。同时最后空集产生的位置与第二个互补单元子句产生的位置相差距离也得到了明显的缩短，说明“1.采用单元优先策略 ”对判断结束有一定的提升效果，但还是不能在互补子句一产生就直接进行判断结束，还有一定的提升空间。

3.仅采用”2.循环结束判断“优化

1. 实验结果展示示例（可图可表可文字，尽量可视化）

实验任务1 Aipine Club

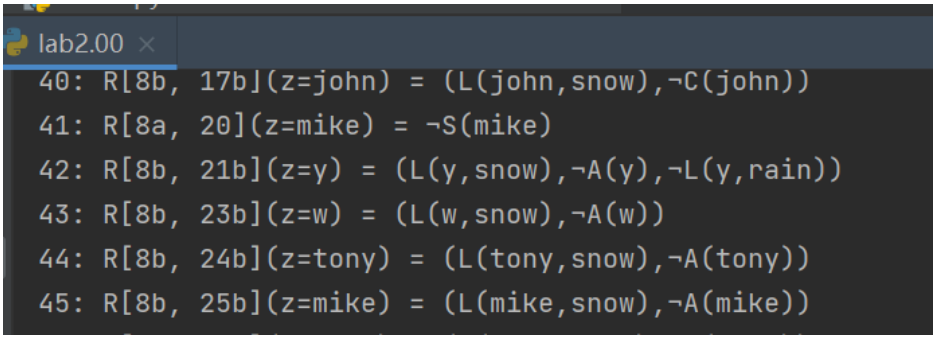


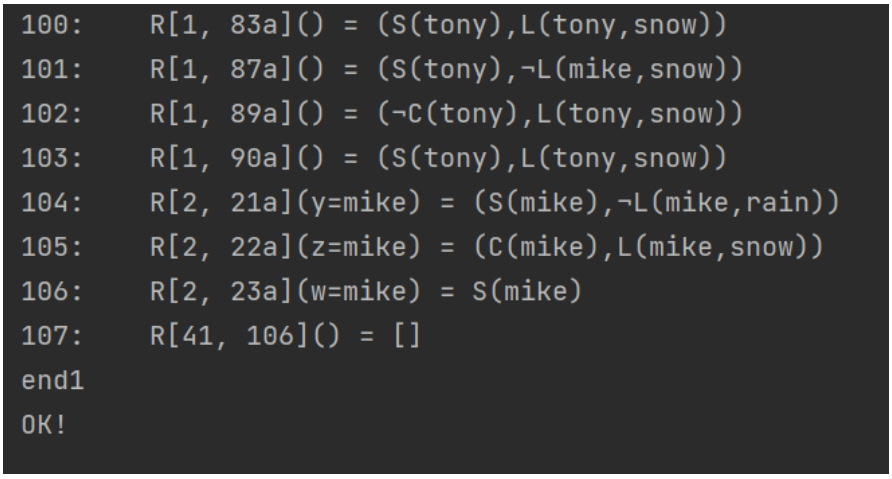
实验任务2 Graduate Student

```
lab2.00
D:\Anaconda\python.exe D:/桌面文件/python_project/lab2.00.py
Please enter a clause set
GradStudent(sue)
(¬GradStudent(x), Student(x))
(¬Student(x),HardWorker(x))
¬HardWorker(sue)

GradStudent(sue)
(¬GradStudent(x),Student(x))
(¬Student(x),HardWorker(x))
¬HardWorker(sue)
5:  R[1, 2a](x=sue) = Student(sue)
6:  R[2b, 3a]() = (¬GradStudent(x),HardWorker(x))
7:  R[3b, 4](x=sue) = ¬Student(sue)
8:  R[5, 7]() = []
end1
OK!
```

实验任务3 Block World

```
lab2.00
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))
12: R[1, 6a](x=tony) = (S(tony),C(tony))
13: R[1, 11a](w=tony) = (¬C(tony),S(tony))
14: R[2, 6a](x=mike) = (S(mike),C(mike))
15: R[2, 11a](w=mike) = (¬C(mike),S(mike))
16: R[3, 6a](x=john) = (S(john),C(john))
17: R[3, 11a](w=john) = (¬C(john),S(john))
18: R[4, 7b](y=tony) = ¬C(tony)
19: R[4, 9a](u=rain) = ¬L(mike,rain)
20: R[5, 9a](u=snow) = ¬L(mike,snow)
```





2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

评测指标：

最后两个互补的单元子句产生的位置及最终空子句产生的位置，三项数据可由最后空集产生所在行得知，如：

28: R[26,27]（）= [] 得到指标（26，27，28）

3个任务的评测指标分别为：（9，11，12）（5，7，8）（41，106，107）

分析：

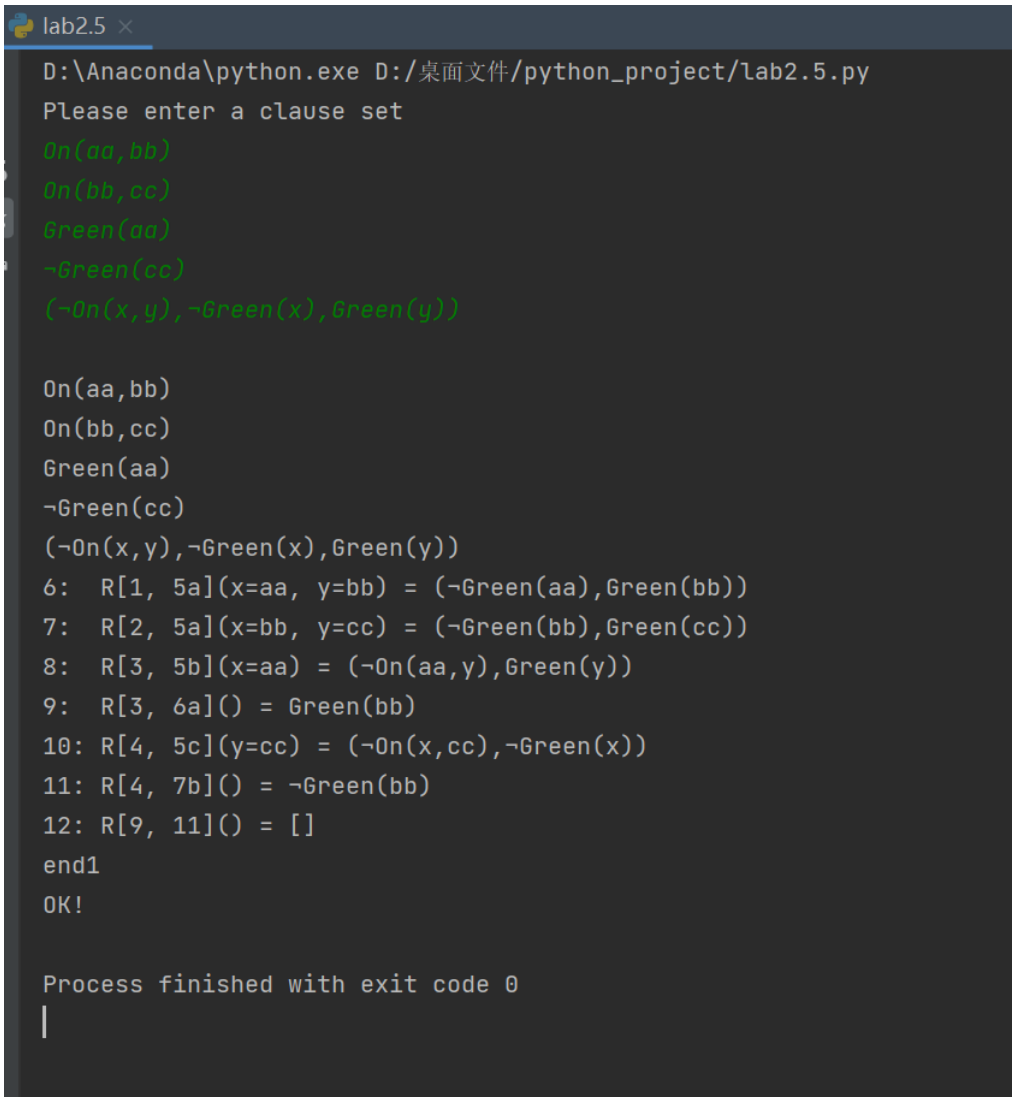
将评测指标与无优化的评测指标进行对比：（3，19，25）（4，8，10）（5，111，132）

显然3个任务最后产生空子句的位置都得到提前，能在互补子句一产生就直接进行判断结束，同时还能更早地发现有互补子句的产生，说明使用”2.循环结束判断“优化对于判断结束有着较好的效果；但对于互补子句的产生的没有效果，只能更早地发现有互补子句的产生，这里还有一定的提升空间。

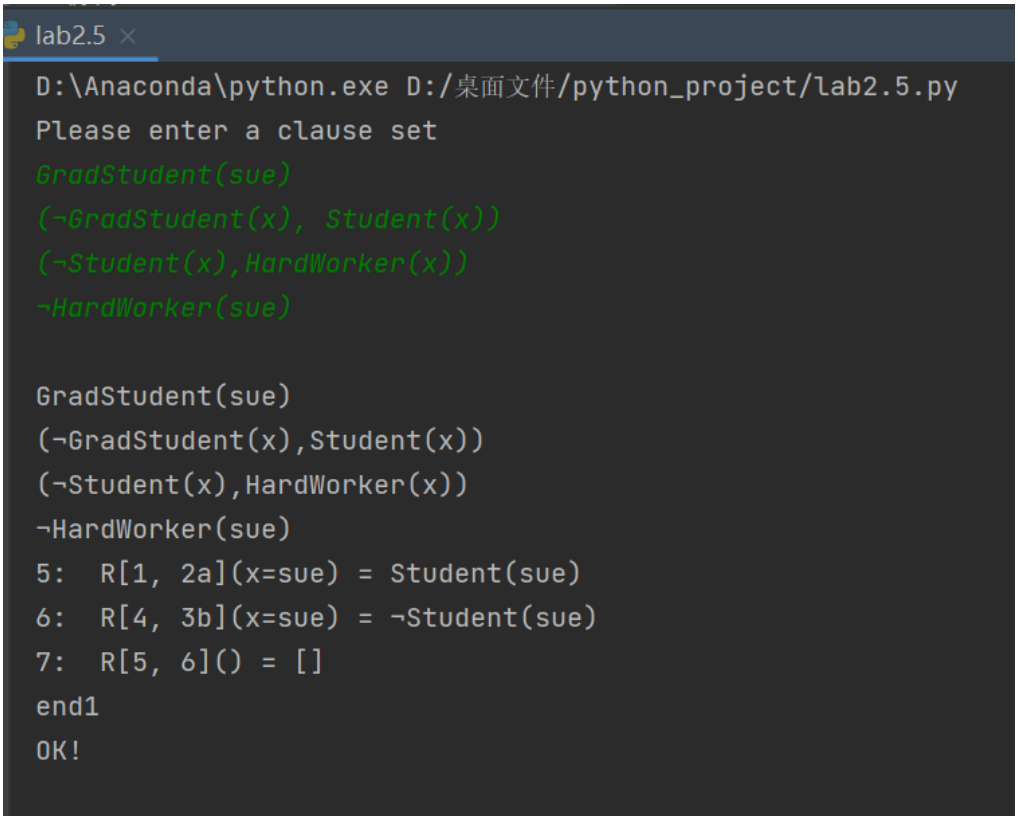
4.同时采用“1.采用单元优先策略 ”和”2.循环结束判断“优化

1. 实验结果展示示例（可图可表可文字，尽量可视化）

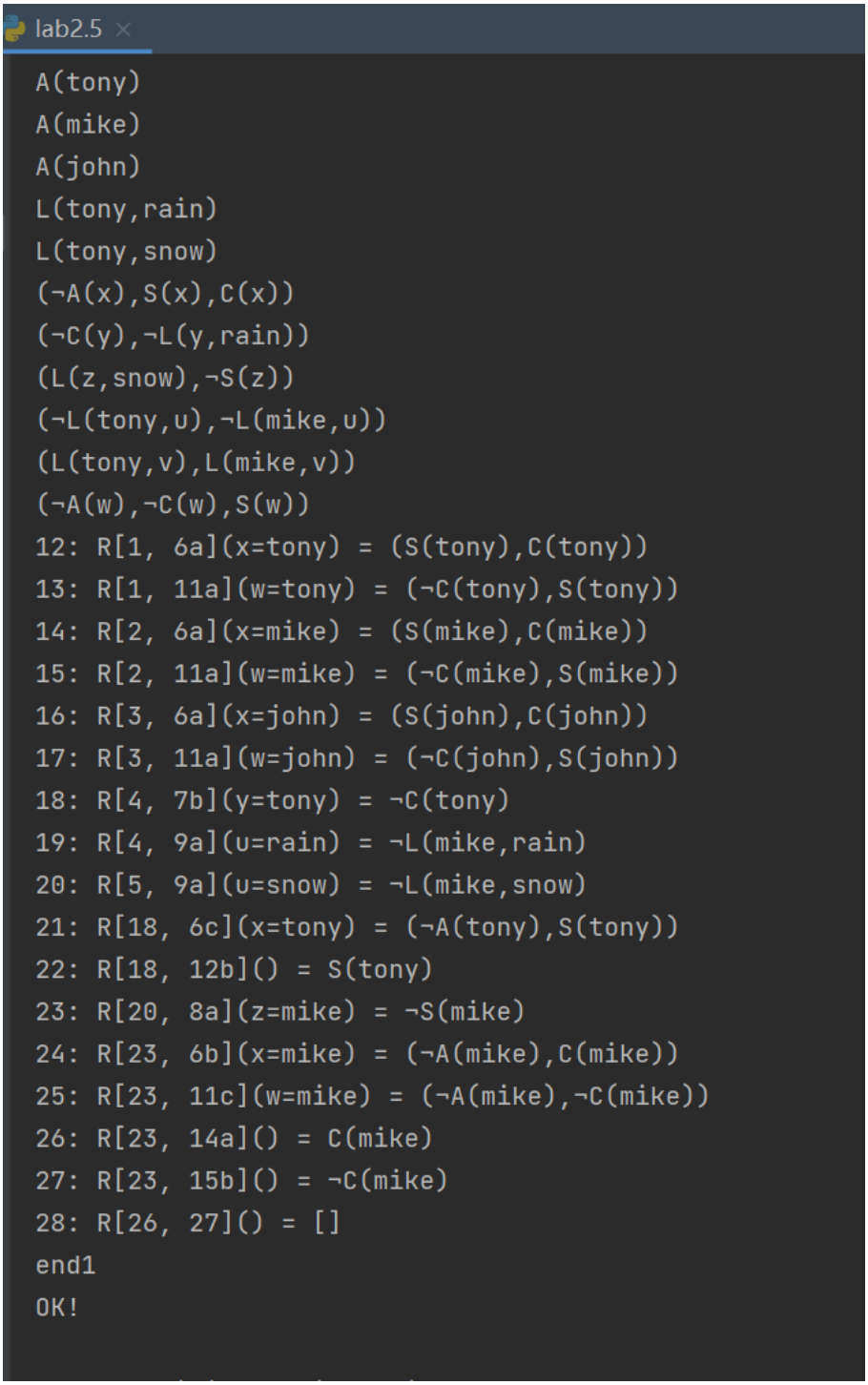
实验任务1 Aipine Club



实验任务2 Graduate Student



实验任务3 Block World



2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

评测指标：

最后两个互补的单元子句产生的位置及最终空子句产生的位置，三项数据可由最后空集产生所在行得知，如：

28: R[26,27]（）= [] 得到指标（26，27，28）

3个任务的评测指标分别为：（9，11，12）（5，6，7）（26，27，28）

分析：

将评测指标前三次的评测指标进行对比，前三次3个任务的评测指标分别为：

1. （3，19，25） （4，8，10） （5，111，132）
2. （9，11，16） （5，6，8） （26，27，31）
3. （9，11，12） （5，7，8） （41，106，107）

可知与前三次进行对比，本次互补子句产生的位置及最后空子句的位置都是最靠前的，说明第四次同时采用“1.采用单元优先策略 ”和”2.循环结束判断“优化，在互补子句的产生及判断结束方面都取得了四次中最好的效果。

四、 思考题

本次实验无思考题。

五、 参考资料

1. 实验文档：实验2-归结原理.pdf
2. 课本：《人工智能》(第三版) 清华大学出版社
3. 课件：ch2 知识表示和推理 II.pptx