1 命题逻辑的归结推理 (不含谓词)

1.1 算法原理

首先基于带剪枝的暴力搜索, 对命题条件进行归结推理至推出空子句, 得到一系列推理过程:

Algorithm 1: 命题逻辑的归结推理 (不含谓词)

初始化总子句集 S
取总子句集 S 的最后一个子句加入支持集 T
while True **do**
    初始化新增子句集 $N = \varnothing$
    将已有子句集 S 按照子句的长度从短到长排序
    for Clause1 $\in S$ **do**
        for Clause2 $\in S$ **do**
            if Clause1 = Clause2 **或** Clause1 与 Clause2 已归结过 **或** (Clause1 $\notin T$ 且 Clause2 $\notin T$) **then**
                continue
            end if
            for Literal1 $\in$ Clause1 **do**
                for Literal2 $\in$ Clause2 **do**
                    if Literal1 与 Literal2 互补 **then**
                        归结推理 (Clause1, Literal1, Clause2, Literal2) 得到新子句 C
                        if C 已出现过 **then**
                            continue
                        end if
                        将得到的推理过程加入 Result
                        将新子句 C 加入新增子句集 N
                        if $C = \varnothing$ **then**
                            break ▷ 推理结束
                        end if
                    end if
                end for
            end for

end for
end for
$S = S \cup N$
$T = T \cup N$
end while

此时得到的推理过程 Result 中有许多重复赘余的推理, 对推理结果没有贡献, 需要从推理结果出发, 根据它们的父子句逐层往上查找, 最终得到最简推理过程:

Algorithm 2: 对推理的化简

初始化有用子句集 $U = \varnothing$
将 Result 中得到的最后一个子句（即空子句）加入队列 Q
while Q 不为空 **do**
  获取队首子句 C
  if C 未遍历过 **then**
    $U = U \cup \{C\}$
    得到推理出 C 的两个父子句 $Fa1$, $Fa2$
    if $Fa1$ 属于支持集 T **then**
      将 $Fa1$ 加入队列 Q
    end if
    if $Fa2$ 属于支持集 T **then**
      将 $Fa2$ 加入队列 Q
    end if
  end if
end while
得到化简后的推理过程, 即有用子句集 U 中的子句
对子句重新编号, 得到最终推理过程 $UsefulResult$

1.2 关键代码展示

首先是归结推理过程:

```
def resolution(KB):
ALL=KB.copy()
support_list=[(ALL[-1],len(ALL)-1)]  #初始化支持集为KB的最后一条
```

```
result=[]    #最终的推理过程
vis=set()    #记忆化,同一子句对只归结一次

newall=[]    #记录各子句的最初编号并对它们按照子句长度进行排序
for i,x in enumerate(ALL,0):
    newall.append((x,i))
idx=len(newall)
while True:
    newclauset=[] #新增子句集
    newall=sorted(newall,key=lambda x:len(x[0]))
    for clause1,clause1_idx in newall:
        for clause2,clause2_idx in newall:
            if clause1_idx==clause2_idx :  #判重
                continue
            if (clause1,clause2) in vis:   #剪枝,同一子句对只归结一
               次
                continue
            sup_list=[x[0] for x in support_list]
            if clause2 not in sup_list and clause1 not in sup_list:
                 #必须至少有一个子句在支持集
                continue
            for literal_idx1 in range(len(clause1)):
                for literal_idx2 in range(len(clause2)):
                    literal1,literal2=clause1[literal_idx1],clause2
                       [literal_idx2]
                    if not iscomplementary(literal1,literal2):
                        continue
                    '''处理互补对'''
                    newclause=resolve(clause1,clause2,literal_idx1,
                       literal_idx2)#归结得到新子句
                    newcset=[x[0] for x in newclauset]
                    if contains_unordered(ALL,newclause) or
                       contains_unordered(newcset,newclause):   #若
                       新子句已出现过则不采纳
```

```
                    continue
                vis.add((clause1,clause2))  #记忆化
                idx1=Index(literal_idx1,clause1_idx,len(clause1
                    ))  #得到子句和具体文字位置的id,如"3b"
                idx2=Index(literal_idx2,clause2_idx,len(clause2
                    ))
                seq=sequence(newclause,idx1,idx2)  #得到推理过
                    程的格式,如"5 R[3b,4b]=('~P',)"
                result.append(seq)

                newclauset.append((newclause,idx))
                idx+=1
                if newclause==():  #归结得到空子句,推理结束
                    return result
    '''更新子句集和支持集'''
    newall.extend(newclauset)
    support_list.extend(newclauset)
```

接着是对推理过程的化简:

```
def simplify(res,size):   #size是初始子句集大小
useful=[]   #有用子句集
que=[len(res)]  #队列,初始时将res中的最后一个空子句加入队列
vis=set()   #记忆,每个子句只搜一次
while que!=[]:
    front=que.pop(0)
    if front in vis:
        continue
    vis.add(front)

    useful.append(res[front-1])   #队首元素加入有用子句集
    fa1,fa2=getfa(res[front-1])   #获得此推理出此子句的双亲子句
    '''只对支持集的子句进行搜索,以免重复输出初始子句集中的子句'''
    if fa1>size:
```

```
            que.append(fa1)
        if fa2>size:
            que.append(fa2)
    useful.reverse()
    usefulres=res[0:size]+useful    #得到最终的简化推理过程
    '''对简化后的推理过程重新编号,得到题目所需的输出格式'''
    for i in range(size,len(usefulres)):
        fa1,fa2=getfa(usefulres[i])
        newnum1=str(newnum(fa1,res,usefulres,size))
        newnum2=str(newnum(fa2,res,usefulres,size))
        #print(fa1,newnum1,fa2,newnum2)
        usefulres[i]=Resequence(usefulres[i],str(fa1),str(fa2),newnum1,
            newnum2)
    return usefulres
```

1.3 创新点 & 优化

由于 ppt 中给出的样例过于简单, 无法很好的测试算法的正确性, 因此我多创造了几个复杂的样例进行测试, 并根据问题对算法进行了优化, 以下是优化点:

- 搜索剪枝, 每个子句对只归结一次, 生成的新子句必须唯一
- 为了尽可能快的推理中空子句, 每次搜索前将子句集按照子句长度由短到长进行排序, 推理时优先归结短的子句。
- 简化推理过程: 每次归结的两个子句必须至少有一个处于支持集中

1.4 实验结果展示

由于 ppt 中的测试样例过于简单, 于是我自行生成了多几个复杂的样例测试算法的正确性, 以下是结果展示:

样例一:PPT 中的样例:

```
KB = [
    ('FirstGrade',),
    ('~FirstGrade','Child'),
    ('~Child',)
]
```

```
1 ('FirstGrade',)
2 ('~FirstGrade', 'Child')
3 ('~Child',)
4 R[3,2b]=('~FirstGrade',)
5 R[1,4]=()
```

样例二: 四子句矛盾

```
KB = [
('P', 'Q'),
('P', '~Q'),
('~P', 'Q'),
('~P', '~Q')
]
```

```
1 ('P', 'Q')
2 ('P', '~Q')
3 ('~P', 'Q')
4 ('~P', '~Q')
5 R[3b,4b]=('~P',)
6 R[5,1a]=('Q',)
7 R[2a,4a]=('~Q',)
8 R[7,6]=()
```

样例三: 长链推理

```
132 if __name__=='__main__':
133     KB = [
134     ('X', 'Y'),
135     ('~Y', 'Z'),
136     ('~X', 'W'),
137     ('~W', 'V'),
138     ('~V',),
139     ('~Z',)
140 ]
```

```
1 ('X', 'Y')
2 ('~Y', 'Z')
3 ('~X', 'W')
4 ('~W', 'V')
5 ('~V',)
6 ('~Z',)
7 R[6,2b]=('~Y',)
8 R[7,1b]=('X',)
9 R[8,3a]=('W',)
10 R[9,4a]=('V',)
11 R[5,10]=()
```

样例四: 大型组合矛盾

```
132 if __name__=='__main__':
133     KB = [
134     ('A', 'B'),
135     ('~A', 'C'),
136     ('~B', 'D'),
137     ('~C', 'E'),
138     ('~D', 'F'),
139     ('~E',),
140     ('~F',)
141 ]
```

```
1 ('A', 'B')
2 ('~A', 'C')
3 ('~B', 'D')
4 ('~C', 'E')
5 ('~D', 'F')
6 ('~E',)
7 ('~F',)
8 R[7,5b]=('~D',)
9 R[8,3b]=('~B',)
10 R[9,1b]=('A',)
11 R[10,2a]=('C',)
12 R[11,4a]=('E',)
13 R[6,12]=()
```

2 最一般合一算法

2.1 算法原理

算法的原理如下。若差异元素其中一个为变量, 则检查循环依赖后即可替换; 若两者都是谓词, 则将谓词拆解为谓词名与参数列表, 递归计算 MGU。具体执行过程如下:

Algorithm 3: 最一般合一算法

初始化合一字典 $\theta \leftarrow \emptyset$
将 $\text{zip}(\text{atom}_1, \text{atom}_2)$ 加入队列 Q
while Q 不为空 **do**
    取出 (s, t)
    通过 θ 映射 s 和 t 为最新值
    if $s = t$ **then**
        continue
    end if
    if s 是变量 **then**
        if s 与 t 存在循环依赖 **then**
            返回失败
        end if
        $\theta(s) \leftarrow t$
        continue
    end if
    if t 是变量 **then**
        if t 与 s 存在循环依赖 **then**
            返回失败
        end if
        $\theta(t) \leftarrow s$
        continue
    end if
    if s 和 t 均为复合项 **then**
        提取 s 和 t 的谓词名和参数
        if 谓词名不同 **then**
            返回失败

27: **end if**
28: 将 s 和 t 的对应参数两两配对加入 Q
29: **continue**
30: **end if**
31: 返回失败
32: **end while**
33: 更新 θ 使其映射为最新值
34: 返回 θ

3 关键代码展示

对复合项的处理: 将其拆解为谓词名 + 参数列表:

```
def Split(x):
args=[]
start=x.find('(')
func_name=x[:start]
args_str=x[start+1:-1]
s=''
dep=0
for c in args_str:  #利用栈的思想,提取谓词的第一层参数
    if c=='(':
        dep+=1
    elif c==')':
        dep-=1
    if c==',' and dep==0:
        args.append(s)
        s=''
    else:
        s+=c
if s:
    args.append(s.strip())
return func_name,args    #返回谓词名和参数列表
```

将替换的映射应用于原子公式 term:

```
def Map(term,dictionary):
if not dictionary:
    return term
if isvariable(term):
    if term in dictionary:
        return Map(dictionary[term],term)
    else:
        return term
elif isfunc(term):   #处理符合项:递归替换它们的参数
    func_name,args=Split(term)
    new_args=[Map(arg,dictionary) for arg in args]
    return f'{func_name}({','.join(new_args)})'
else:
    return term
```

检查循环依赖:

```
def occurs_check(v,term):#检查变量v是否在term中出现
if v==term:
    return True
if isfunc(term):
    _,args=Split(term)
    for t in args:
        if occurs_check(v,t):
            return True
return False
```

对两个原子公式列表应用于 MGU 算法:

```
def MGU(para1,para2):
if len(para1) != len(para2):
    return None
```

```
subst={}
equa=list(zip(para1,para2))
while equa:
    s,t=equa.pop(0)
    s=Map(s,subst)    #更新s和t的最新值
    t=Map(t,subst)
    if s==t:
        continue
    if isvariable(s):
        if occurs_check(s,t):
            return None
        subst[s]=t
        continue
    if isvariable(t):
        if occurs_check(t,s):
            return None
        subst[t]=s
        continue
    if isfunc(s) and isfunc(t):
        s_name,s_args=Split(s)
        t_name,t_args=Split(t)
        if s_name != t_name or len(s_args)!=len(t_args):
            return None
        equa.extend(zip(s_args,t_args))
        continue
    return None    #不满足上述四个if条件,则存在不可替换的两个常量,返
        回失败
for x in list(subst.keys()):
    subst[x]=Map(subst[x],subst)  #更新每个变量的替换至最新
return subst
```

3.1 创新点 & 优化

- 利用栈的思想, 提取谓词的参数

- 优化了课本上的 mgu 算法过程, 统一对两个原子公式的参数进行分析, 在队列中更新参数值
- 检查了循环依赖项, 这在课本与 ppt 中的样例并无体现

3.2 实验结果展示

样例一二:PPT 中的样例

```
para1 = ['P(xx,a)']
para2 = ['P(b,yy)']
```

```
PS D:\college\AI实验\homework
{'xx': 'b', 'yy': 'a'}
```

```
para1 = ['P(a,xx,f(g(yy)))']
para2 = ['P(zz,f(zz),f(uu))']
```

```
PS D:\college\AI实验\homework\week3-4> & D:/
{'zz': 'a', 'xx': 'f(a)', 'uu': 'g(yy)'}
```

样例三: 存在循环依赖

```
para1 = ['x']
para2 = ['f(x,y)']
```

```
PS D:\colleg
None
```

样例四: 无法合一的情况

```
para1 = ['P(f(x), x)']
para2 = ['P(f(a), b)']
```

```
PS D:\college\AI实
None
```

4 命题逻辑的归结推理 (谓词的情况)

4.1 算法原理

本算法大体流程与" 命题逻辑的归结推理 (不含谓词的情况)" 大体一致, 不同点是在得到互补文字后需要判断它们是否可以进行 mgu 合一:
若合一失败, 则不能进行归结
若合一成功, 则先将它们合一, 再通过归结得到新子句

4.2 关键代码展示

与命题逻辑的归结推理 (不含谓词的情况) 不同的是, 本算法需要另外实现 Sub 函数, 即将 mgu 应用于替换子句:

```
def sub(clause,dictionary):
```

```
newclause=[]
for x in clause:
    newclause.append(Map(x,dictionary))
return tuple(newclause)
```

主要推理部分的代码如下, 新增了关于 mgu 的处理

```
def resolution(KB):
ALL=list(KB)
support_list=[ALL[-1]]
result=[]
vis=set()
while True:
    newclauset=[]
    for clause1_idx in range(len(ALL)):
        for clause2_idx in range(clause1_idx+1,len(ALL)):
            if clause1_idx==clause2_idx :
                continue
            clause1,clause2=ALL[clause1_idx],ALL[clause2_idx]
            if (clause1,clause2) in vis:
                continue
            if clause2 not in support_list and clause1 not in
               support_list:
                continue
            for literal_idx1 in range(len(clause1)):
                for literal_idx2 in range(len(clause2)):
                    literal1,literal2=clause1[literal_idx1],clause2
                       [literal_idx2]
                    if not iscomplementary(literal1,literal2):
                        continue
                    '''处理互补对'''
                    literal1=literal1.replace('~','')
                    literal2=literal2.replace('~','')
                    literal1,literal2=[literal1],[literal2]
```

```
                    mgu_dict=MGU(literal1,literal2)
                    if mgu_dict==None:  #若不能进行合一,则不可归结
                        continue
                    mgu_clause1=sub(clause1,mgu_dict)
                    mgu_clause2=sub(clause2,mgu_dict)
                    newclause=resolve(mgu_clause1,mgu_clause2,
                       literal_idx1,literal_idx2)
                    if newclause in ALL or newclause in newclauset:
                        continue
                    vis.add((clause1,clause2))
                    idx1=Index(literal_idx1,clause1_idx,len(clause1
                       ))
                    idx2=Index(literal_idx2,clause2_idx,len(clause2
                       ))
                    seq=sequence(newclause,idx1,idx2,mgu_dict)
                    result.append(seq)
                    newclauset.append(newclause)
                    if newclause==():
                        return result
    ALL.extend(newclauset)
    support_list.extend(newclauset)
```

4.3 实验结果展示

样例一二三:PPT 中的样例

```
KB1=[('GradStudent(sue)',),('~GradStudent(sue)','Student(x)'),('~Student(x)','Hardworker(x)'),
('~Hardworker(sue)',)]
```

```
1 ('GradStudent(sue)',)
2 ('~GradStudent(sue)', 'Student(x)')
3 ('~Student(x)', 'Hardworker(x)')
4 ('~Hardworker(sue)',)
5 R[3b,4]{x=sue}=('~Student(sue)',)
6 R[2b,5]{x=sue}=('~GradStudent(sue)',)
7 R[1,6]=()
```

```
KB2=[('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)')]
```

```
PS D:\college\AI实验\homework\week3-4> & D:/Anaconda3/envs/davi
1 ('A(tony)',)
2 ('A(mike)',)
3 ('A(john)',)
4 ('L(tony,rain)',)
5 ('L(tony,snow)',)
6 ('~A(x)', 'S(x)', 'C(x)')
7 ('~C(y)', '~L(y,rain)')
8 ('L(z,snow)', '~S(z)')
9 ('~L(tony,u)', '~L(mike,u)')
10 ('L(tony,v)', 'L(mike,v)')
11 ('~A(w)', '~C(w)', 'S(w)')
12 R[6c,11b]{x=w}=('~A(w)', 'S(w)')
13 R[8b,12b]{z=w}=('L(w,snow)', '~A(w)')
14 R[9a,13a]{w=tony}{u=snow}=('~L(mike,snow)', '~A(tony)')
15 R[2,13b]{w=mike}=('L(mike,snow)',)
16 R[15,14a]=('~A(tony)',)
17 R[1,16]=()
```

```
KB3=[('On(tony,mike)',),('On(mike,john)',),('Green(tony)',),('~Green(john)',),('~On(xx,yy)','~Green
(xx)','Green(yy)')]
```

```
PS D:\college\AI实验\homework\week3-4> & D:/Anaconda
1 ('On(tony,mike)',)
2 ('On(mike,john)',)
3 ('Green(tony)',)
4 ('~Green(john)',)
5 ('~On(xx,yy)', '~Green(xx)', 'Green(yy)')
6 R[4,5c]{yy=john}=('~On(xx,john)', '~Green(xx)')
7 R[3,5b]{xx=tony}=('~On(tony,yy)', 'Green(yy)')
8 R[2,6a]{xx=mike}=('~Green(mike)',)
9 R[1,7a]{yy=mike}=('Green(mike)',)
10 R[9,8]=()
```

样例四五: 额外生成的, 谓词层次较深, 替换较复杂的样例:

```
    KB4 = [
('P(a,f(b))',), #1
('Q(c,y)',), #2
('~P(x,z)', '~Q(u,v)', 'R(x,u,f(z))'), #3
('~R(a,c,w)', 'S(w,b)'), #4
('~S(f(v),b)', 'T(v)'), #5
('~T(f(b))',), #6
('R(a,c,f(f(b)))',) #7
]
```

```
PS D:\college\AI实验\homework\week3-4> &
1 ('P(a,f(b))',)
2 ('Q(c,y)',)
3 ('~P(x,z)', '~Q(u,v)', 'R(x,u,f(z))')
4 ('~R(a,c,w)', 'S(w,b)')
5 ('~S(f(v),b)', 'T(v)')
6 ('~T(f(b))',)
7 ('R(a,c,f(f(b)))',)
8 R[4a,7]{w=f(f(b))}=('S(f(f(b)),b)',)
9 R[5a,8]{v=f(b)}=('T(f(b))',)
10 R[6,9]=()
```

```
    KB5 = [
('F(f(a),g(b))',), #1
('G(h(c),k(d))',), #2
('~F(x,y)', '~G(u,v)', 'S(x,u)'), #3
('~S(w,z)', 'T(w,z)'), #4
('~T(f(a),h(c))',), #5
('S(f(a),h(c))',) #6
]
```

```
PS D:\college\AI实验\homework\week3-4> & D:/
1 ('F(f(a),g(b))',)
2 ('G(h(c),k(d))',)
3 ('~F(x,y)', '~G(u,v)', 'S(x,u)')
4 ('~S(w,z)', 'T(w,z)')
5 ('~T(f(a),h(c))',)
6 ('S(f(a),h(c))',)
7 R[4a,6]{w=f(a)}{z=h(c)}=('T(f(a),h(c))',)
8 R[5,7]=()
```