

(2022 学年春季学期)


教学班级	202320346	专业（方向）	计算机科学与技术
学号	21312450	姓名	林隽哲

课堂中已经证明了定理：两个子句 C_1 和 C_2 的归结式 C 是 C_1 和 C_2 的逻辑推论。由此我们得到了推论：子句集 $S=\{C_1, C_2, \dots, C_n\}$ 与子句集 $S_1=\{C, C_1, C_2, \dots, C_n\}$ 的不可满足性是等价的（其中 C 是 C_1 和 C_2 的归结式）。同时在课堂上也证明了归结推理的合理性（定理：如果 $S \vdash C$ ，那么 $S \vdash ()$ ）与完备性（定理： $S \vdash ()$ ，当且仅当 $S \vdash ()$ ，当且仅当 S 不可满足）。由上述理论支持，可以将一阶逻辑的归结推理过程总结如下：

命题逻辑中，若给定前提集 F 和命题集 R ，则：

- (1) 把 F 转化成子句集表示，得到子句集 S_0 ;
- (2) 把命题 R 的否定式 $\neg R$ 也转化成子句集表示，并将其加到 S_0 中，得 $S = S_0 \cup S_{\neg R}$;
- (3) 对子句集 S 反复应用归结推理规则（推导），直至导出含有空子句的扩大子句集为止。即出现归结式为空子句时，表明已找到矛盾，证明过程结束。

在归结过程中，若 S 中两个子句间有相同互补文字的谓词，但它们的项不同，则必须找出对应的不一致项，并对其进行变量置换，使它们的对应项一致，然后再尝试用它们推导出空子句。通常可以使用最一般合一算法对两个原子公式进行合一，如下：

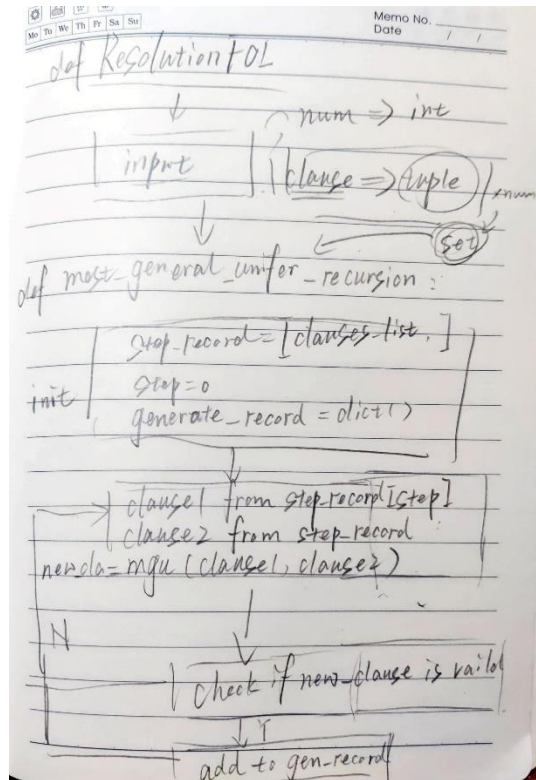
 最一般合一算法：

- 输入：两个原子公式，它们具有相同的谓词，不同的参数项和 “ \neg ”
- 输出：一组变量替换/赋值
- 算法流程：
 - $k = 0; \sigma_0 = \{\}; S_0 = \{f, g\}$
 - 如果 S_k 中的公式等价，返回 σ_k 作为最一般合一的结果
 - 否则找出 S_k 中的不匹配项 $D_k = \{e_1, e_2\}$
 - 如果 $e_1 = V$ 是变量， $e_2 = t$ 是一个不包含变量 V 的项，将 “ $V = t$ ” 添加到赋值集合 $\sigma_{k+1} = \sigma_k \cup \{V = t\}$ ；并将 S_k 中的其它 V 变量也赋值为 t ，得到 S_{k+1} ；
 $k = k + 1$ ，转到第二步
 - 否则合一失败

Tips: 变量替换是从两个原子公式中找到的，但是最后要施加给整个子句的，

伪代码

这里给出粗略的伪代码实现：



不难看出我使用的是一种分步骤递推的方法进行推理。其中，当 step_record 的下一递推为空集时说明推理失败，而当 new_clause 为空集时说明找到了一种成功的递推方法，剩下的就是将保存的推理过程打印出来即可。

关键代码展示（带注释）

```
1 while True:
2     step_record.append([])
3     step += 1
4
5     for i in range(len(step_record[step-1])):
6         for s in range(step):
7             for j in range(len(step_record[s])):
8                 # find a pair of clauses to resolve
9                 clause1 = step_record[step-1][i]
10                clause2 = step_record[s][j]
11                for m in range(len(clause1)):
12                    for n in range(len(clause2)):
13                        atomic_rule1 = clause1[m]
14                        atomic_rule2 = clause2[n]
15                        if not ((atomic_rule1.startswith('~') ^ atomic_rule2.startswith('~')) or
16                              (atomic_rule1.startswith('.') ^ atomic_rule2.startswith('.'))):
17                            continue
18                        try:
19                            mgu_res = most_general_unifier(atomic_rule1, atomic_rule2)
20                        except UnificationError:
21                            continue
22                        new_clause = list(clause1 + clause2)
23                        new_clause.remove(atomic_rule1)
24                        new_clause.remove(atomic_rule2)
25
26                        # substitute the variable in the clause
27                        for var, value in mgu_res.items():
28                            new_clause = [re.sub(r'\b' + var + r'\b', value, clause) for clause in new_clause]
29
30                        # remove duplicate
31                        new_temp = list(set(new_clause))
32                        new_temp.sort(key=lambda x: new_clause.index(x))
33                        new_clause = tuple(new_temp)
34
35                        # check if the new clause is already in the record
36                        for k in range(step+1):
37                            for l in range(len(step_record[k])):
38                                if step_record[k][l] == new_clause:
39                                    break
40                            else:
41                                continue
42                        break
43                    else: # if the new clause is not in the record
44                        step_record[step].append(new_clause)
45                        generate_record[new_clause] = (step-1, i, s, j, n, mgu_res)
46
47                        # if the new clause is an empty clause, then the resolution is done
48                        if new_clause == ():
49                            resolution_path = []
50                            temp_queue = [[]]
51                            while temp_queue:
52                                temp = temp_queue.pop(0)
53                                if temp in generate_record:
54                                    s1, i, s2, j, n, mgu_res = generate_record[temp]
55                                    parent_clause1 = step_record[s1][i]
56                                    parent_clause2 = step_record[s2][j]
57                                    temp_queue.append(parent_clause1)
58                                    temp_queue.append(parent_clause2)
59
60                                # the way to print the resolution path, you can adjust it by yourself
61                                parent_clause1_str = ', '.join([f'\033[92m{clause}\033[0m' if l != n else f'\033[91m{clause}\033[0m' for l, clause in enumerate(parent_clause1)])
62                                parent_clause2_str = ', '.join([f'\033[92m{clause}\033[0m' if j != n else f'\033[91m{clause}\033[0m' for j, clause in enumerate(parent_clause2)])
63                                var_assign_str = ', '.join([f'\033[94m{key} = {value}\033[0m' for key, value in mgu_res.items()])
64                                temp_str = f'\033[93m{temp}\033[0m'
65                                resolution_path.append(f'({parent_clause1_str}) + ({parent_clause2_str}) [{var_assign_str}] => ({temp_str})')
66
67                            resolution_path = resolution_path[::-1]
68
69                            for clause in resolution_path:
70                                print(clause)
71                            return
72
73                        if step_record[step] == []:
74                            print('Do not have a resolution path.')
75                            return
```

上图是归结过程的核心代码，这里简单的采用了暴力搜索的方式寻找正确的递推路径。相关的实现细节以及说明可见代码内的注释。

```

1  def most_general_unifier_recursion(atomic_rule1: str, atomic_rule2: str) -> dict:
2
3      """Return the most general unifier of two atomic rules.
4      Parameters:
5          atomic_rule1: str
6          atomic_rule2: str
7
8      Return:
9          unifier: dict
10     """
11
12     import re
13
14     # term:
15     # - a variable or a constant is a term
16     # - if t1, t2, ..., tn are terms, and f is a function symbol of arity n, then f(t1, t2, ..., tn) is a term
17     # - cause we only concern with first-order Logic, predicate is not a term
18
19     # variable naming convention:
20     # - variables can only use names defined in vars
21
22     # constant naming convention:
23     # - start with a lowercase letter
24     # - can only contain letters, numbers, and underscores
25
26     # function naming convention:
27     # - start with a lowercase letter
28     # - can only contain letters, numbers, and underscores
29     # - one or more terms separated by commas and enclosed in parentheses behind the function name
30
31     # predicate naming convention:
32     # - start with a capital letter
33     # - can only contain letters, numbers, and underscores
34     # - one or more terms separated by commas and enclosed in parentheses behind the predicate name
35     # sample: P(x, y), Q(f(x), g(y, z))
36
37     # Strictly speaking, the regular expression here is not a strict predicate form,
38     # because in the specification, the predicate is started with a capital letter
39     # But for the convenience of the subsequent recursive implementation of the function,
40     # the definition of the predicate is relaxed here
41     pattern = re.compile(r'[~]?([A-Za-z][A-Za-z0-9_]*)(\((.*)\))')
42     match1 = pattern.match(atomic_rule1)
43     match2 = pattern.match(atomic_rule2)
44
45     if match1 is None or match2 is None:
46         raise UnificationError('The atomic rules are not in a valid form.')
47
48     if match1.group(1) != match2.group(1):
49         raise UnificationError('The predicates are different.')
50
51     terms1 = [term.strip() for term in match1.group(2).split(',')]
52     terms2 = [term.strip() for term in match2.group(2).split(',')]
53
54     if len(terms1) == 0 or len(terms2) == 0:
55         raise UnificationError('The atomic rules are not in a valid form.')
56
57     if len(terms1) != len(terms2):
58         raise UnificationError('The number of arguments is different.')
59
60     # for more natural handling of variable substitution and lookup, a custom Term class is used here
61     # sure, you can also use regular expressions to implement it,
62     # you can see my implementation in first-order resolution
63     terms1 = [Term(term) for term in terms1]
64     terms2 = [Term(term) for term in terms2]
65
66     unifier = {}
67     error_msg = 'Error: Failed to unify the atomic rules.'
68     for i in range(len(terms1)):
69         if terms1[i] != terms2[i]:
70             var_flg = terms1[i].is_var() + 2*terms2[i].is_var()
71             if var_flg:
72                 var = terms1[i] if var_flg & 1 else terms2[i]
73                 term = terms2[i] if var_flg & 1 else terms1[i]
74                 if var in unifier and unifier[var] != term:
75                     raise UnificationError(error_msg)
76                 if var in term:
77                     raise UnificationError(error_msg)
78                 unifier[var] = term
79             elif terms1[i].is_func() and terms2[i].is_func():
80                 unifier.update(most_general_unifier_recursion(str(terms1[i]), str(terms2[i])))
81             else:
82                 raise UnificationError(error_msg)
83
84     for key, value in unifier.items():
85         for key2, value2 in unifier.items():
86             if key != key2 and key in value2:
87                 unifier[key2] = value2.substitute(key, value)
88
89     return unifier

```

上述是最一般合一算法实现的核心代码，为了方便处理变量的赋值以及蕴含关系的判断，我简单的定义了一个 Term 类，以对谓词中的项进行统一的处理，Term 类的具体实现代码如下：

```
1 class Term:
2
3     __slots__ = ['type', 'name', 'args']
4
5     __vars = ['x', 'y', 'z', 'u', 'v', 'w']
6
7     def __init__(self, term: str) -> NoReturn:
8
9         import re
10
11         self.type = None
12         self.name = None
13         self.args = None
14
15         var_const_pattern = re.compile(r'([a-z][a-z0-9_]+)')
16         function_pattern = re.compile(r'([a-z][a-z0-9_]+)((.*)\\')')
17
18         if function_pattern.match(term):
19             # exm: 'f(g(x, y), z) -> ['f', FuncTerm('g(x, y)'), 'z']
20             self.type = 'func'
21             self.name = function_pattern.match(term).group(1)
22             self.args = [Term(arg) for arg in (arg.strip() for arg in function_pattern.match(term).group(2).split(',') if arg)]
23             if len(self.args) == 0:
24                 raise ValueError('The function has no arguments.')
25         elif var_const_pattern.match(term):
26             self.type = 'var' if term in self.__vars else 'const'
27             self.name = term
28         else:
29             raise ValueError('The term is not in a valid form.')
30
31     def __str__(self) -> str:
32         if self.type == 'var' or self.type == 'const':
33             return self.name
34         elif self.type == 'func':
35             return self.name + '(' + ', '.join([str(arg) for arg in self.args]) + ')'
36
37     def __repr__(self) -> str:
38         return self.__str__()
39
40     def __eq__(self, other: 'Term|str') -> bool:
41
42         if isinstance(other, str):
43             other = Term(other)
44
45         if self.type != other.type:
46             return False
47         if self.type == 'var' or self.type == 'const':
48             return self.name == other.name
49         elif self.type == 'func':
50             if self.name != other.name:
51                 return False
52             if len(self.args) != len(other.args):
53                 return False
54             for i in range(len(self.args)):
55                 if self.args[i] != other.args[i]:
56                     return False
57             return True
58
59     def __ne__(self, other: 'Term|str') -> bool:
60         return not self.__eq__(other)
61
62     def __contains__(self, var: 'Term|str') -> bool:
63
64         if isinstance(var, str):
65             var = Term(var)
66
67         if self.type == 'var' or self.type == 'const':
68             return self == var
69         elif self.type == 'func':
70             for arg in self.args:
71                 if arg.__contains__(var):
72                     return True
73             return False
74
75         return False
76
77     def __hash__(self) -> int:
78         return hash(str(self))
79
80     def is_var(self) -> bool:
81         return self.type == 'var'
82
83     def is_const(self) -> bool:
84         return self.type == 'const'
85
86     def is_func(self) -> bool:
87         return self.type == 'func'
88
89     def substitute(self, var: 'Term|str', term: 'Term|str') -> 'Term':
90         # exm: f = Term('f(x, y)') -> f.substitute('x', 'f(z)') -> Term('f(f(z), y)')
91
92         if isinstance(var, Term):
93             var = str(var)
94         if isinstance(term, Term):
95             term = str(term)
96
97         if self.type == 'const':
98             return self
99         if self.type == 'var':
100             return Term(term) if self.name == var else self
101         if self.type == 'func':
102             return Term(self.name + '(' + ', '.join([str(arg.substitute(var, term)) for arg in self.args]) + ')')
103
```

实验结果及分析

1. 实验结果展示示例

书写测试函数如下：

```
1  def test1():
2      num = 4
3      clause_exm = [
4          'GradStudent(sue)',
5          '(-GradStudent(x), Student(x))',
6          '(-Student(x), HardWorker(x))',
7          '¬HardWorker(sue)'
8      ]
9
10     import io
11     import sys
12
13     print('num:', num)
14     print('clauses:')
15     for clause in clause_exm:
16         print(clause)
17
18     sys.stdin = io.StringIO(f'{num}\n' + '\n'.join(clause_exm) + '\n')
19
20 def test2():
21     num = 11
22     clause_exm = [
23         'A(tony)',
24         'A(mike)',
25         'A(john)',
26         'L(tony, rain)',
27         'L(tony, snow)',
28         '(-A(x), S(x), C(x))',
29         '(-C(y), ¬L(y, rain))',
30         '(L(z, snow), ¬S(z))',
31         '(-L(tony, u), ¬L(mike, u))',
32         '(L(tony, v), L(mike, v))',
33         '(-A(w), ¬C(w), S(w))'
34     ]
35
36     import io
37     import sys
38
39     print('num:', num)
40     print('clauses:')
41     for clause in clause_exm:
42         print(clause)
43
44     sys.stdin = io.StringIO(f'{num}\n' + '\n'.join(clause_exm) + '\n')
```

运行得到测试结果，并将其与预期输出进行对比：

```
-- First Order Logical Resolution --
-----Test 1-----
num: 4
clauses:
GradStudent(sue)
(¬GradStudent(x), Student(x))
(¬Student(x), HardWorker(x))
¬HardWorker(sue)
(¬HardWorker(sue)) + (¬Student(x), HardWorker(x)) [x = sue] => ('¬Student(sue)',)
(GradStudent(sue)) + (¬GradStudent(x), Student(x)) [x = sue] => ('Student(sue)',)
(Student(sue)) + (¬Student(sue)) [] => ()

-----Test 2-----
num: 11
clauses:
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))
(L(tony, snow)) + (¬L(tony, u), ¬L(mike, u)) [u = snow] => ('¬L(mike, snow)',)
(A(mike)) + (¬A(x), S(x), C(x)) [x = mike] => ('S(mike)', 'C(mike)')
(A(mike)) + (¬A(w), ¬C(w), S(w)) [w = mike] => ('¬C(mike)', 'S(mike)')
(¬L(mike, snow)) + (L(z, snow), ¬S(z)) [z = mike] => ('¬S(mike)',)
(¬C(mike), S(mike)) + (S(mike), C(mike)) [] => ('S(mike)',)
(S(mike)) + (¬S(mike)) [] => ()
```

题目1

输入

```
KB = {(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))}
```

输出

```
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 (L(tony, snow)) + (¬L(tony, u), ¬L(mike, u)) [u = snow] => ('¬L(mike, snow)',)
13 (A(mike),) + (¬A(x), S(x), C(x)) [x = mike] => ('S(mike)', 'C(mike)')
14 (A(mike),) + (¬A(w), ¬C(w), S(w)) [w = mike] => ('¬C(mike)', 'S(mike)')
15 (¬L(mike, snow)) + (L(z, snow), ¬S(z)) [z = mike] => ('¬S(mike)',)
16 (¬C(mike), S(mike)) + (S(mike), C(mike)) [] => ('S(mike)',)
17 (S(mike),) + (¬S(mike),) [] => ()
```

题目2

输入

```
KB = {(On(tony, Mike),), (On(mike, John),), (Green(tony),), (¬Green(john),), (On(x, yy), ¬Green(x), Green(yy))}
```

输出

```
1 (On(tony, Mike),)
2 (On(mike, John),)
3 (Green(tony),)
4 (¬Green(john),)
5 (On(x, yy), ¬Green(x), Green(yy))
6 (On(x, yy), John) (On(x, John), ¬Green(x))
7 (On(x, yy), ¬Green(x)) (On(tony, yy), Green(yy))
8 (On(x, yy), ¬Green(x)) (On(mike, John),)
9 (On(x, yy), ¬Green(x)) (On(mike, John),)
10 (On(x, yy), ¬Green(x)) (On(mike, John),)
11 (On(x, yy), ¬Green(x)) (On(mike, John),)
12 (On(x, yy), ¬Green(x)) (On(mike, John),)
13 (On(x, yy), ¬Green(x)) (On(mike, John),)
14 (On(x, yy), ¬Green(x)) (On(mike, John),)
15 (On(x, yy), ¬Green(x)) (On(mike, John),)
16 (On(x, yy), ¬Green(x)) (On(mike, John),)
17 (On(x, yy), ¬Green(x)) (On(mike, John),)
18 (On(x, yy), ¬Green(x)) (On(mike, John),)
19 (On(x, yy), ¬Green(x)) (On(mike, John),)
20 (On(x, yy), ¬Green(x)) (On(mike, John),)
21 (On(x, yy), ¬Green(x)) (On(mike, John),)
22 (On(x, yy), ¬Green(x)) (On(mike, John),)
23 (On(x, yy), ¬Green(x)) (On(mike, John),)
24 (On(x, yy), ¬Green(x)) (On(mike, John),)
25 (On(x, yy), ¬Green(x)) (On(mike, John),)
26 (On(x, yy), ¬Green(x)) (On(mike, John),)
27 (On(x, yy), ¬Green(x)) (On(mike, John),)
28 (On(x, yy), ¬Green(x)) (On(mike, John),)
29 (On(x, yy), ¬Green(x)) (On(mike, John),)
30 (On(x, yy), ¬Green(x)) (On(mike, John),)
31 (On(x, yy), ¬Green(x)) (On(mike, John),)
32 (On(x, yy), ¬Green(x)) (On(mike, John),)
33 (On(x, yy), ¬Green(x)) (On(mike, John),)
34 (On(x, yy), ¬Green(x)) (On(mike, John),)
35 (On(x, yy), ¬Green(x)) (On(mike, John),)
36 (On(x, yy), ¬Green(x)) (On(mike, John),)
37 (On(x, yy), ¬Green(x)) (On(mike, John),)
38 (On(x, yy), ¬Green(x)) (On(mike, John),)
39 (On(x, yy), ¬Green(x)) (On(mike, John),)
40 (On(x, yy), ¬Green(x)) (On(mike, John),)
41 (On(x, yy), ¬Green(x)) (On(mike, John),)
42 (On(x, yy), ¬Green(x)) (On(mike, John),)
43 (On(x, yy), ¬Green(x)) (On(mike, John),)
44 (On(x, yy), ¬Green(x)) (On(mike, John),)
45 (On(x, yy), ¬Green(x)) (On(mike, John),)
46 (On(x, yy), ¬Green(x)) (On(mike, John),)
47 (On(x, yy), ¬Green(x)) (On(mike, John),)
48 (On(x, yy), ¬Green(x)) (On(mike, John),)
49 (On(x, yy), ¬Green(x)) (On(mike, John),)
50 (On(x, yy), ¬Green(x)) (On(mike, John),)
51 (On(x, yy), ¬Green(x)) (On(mike, John),)
52 (On(x, yy), ¬Green(x)) (On(mike, John),)
53 (On(x, yy), ¬Green(x)) (On(mike, John),)
54 (On(x, yy), ¬Green(x)) (On(mike, John),)
55 (On(x, yy), ¬Green(x)) (On(mike, John),)
56 (On(x, yy), ¬Green(x)) (On(mike, John),)
57 (On(x, yy), ¬Green(x)) (On(mike, John),)
58 (On(x, yy), ¬Green(x)) (On(mike, John),)
59 (On(x, yy), ¬Green(x)) (On(mike, John),)
60 (On(x, yy), ¬Green(x)) (On(mike, John),)
61 (On(x, yy), ¬Green(x)) (On(mike, John),)
62 (On(x, yy), ¬Green(x)) (On(mike, John),)
63 (On(x, yy), ¬Green(x)) (On(mike, John),)
64 (On(x, yy), ¬Green(x)) (On(mike, John),)
65 (On(x, yy), ¬Green(x)) (On(mike, John),)
66 (On(x, yy), ¬Green(x)) (On(mike, John),)
67 (On(x, yy), ¬Green(x)) (On(mike, John),)
68 (On(x, yy), ¬Green(x)) (On(mike, John),)
69 (On(x, yy), ¬Green(x)) (On(mike, John),)
70 (On(x, yy), ¬Green(x)) (On(mike, John),)
71 (On(x, yy), ¬Green(x)) (On(mike, John),)
72 (On(x, yy), ¬Green(x)) (On(mike, John),)
73 (On(x, yy), ¬Green(x)) (On(mike, John),)
74 (On(x, yy), ¬Green(x)) (On(mike, John),)
75 (On(x, yy), ¬Green(x)) (On(mike, John),)
76 (On(x, yy), ¬Green(x)) (On(mike, John),)
77 (On(x, yy), ¬Green(x)) (On(mike, John),)
78 (On(x, yy), ¬Green(x)) (On(mike, John),)
79 (On(x, yy), ¬Green(x)) (On(mike, John),)
80 (On(x, yy), ¬Green(x)) (On(mike, John),)
81 (On(x, yy), ¬Green(x)) (On(mike, John),)
82 (On(x, yy), ¬Green(x)) (On(mike, John),)
83 (On(x, yy), ¬Green(x)) (On(mike, John),)
84 (On(x, yy), ¬Green(x)) (On(mike, John),)
85 (On(x, yy), ¬Green(x)) (On(mike, John),)
86 (On(x, yy), ¬Green(x)) (On(mike, John),)
87 (On(x, yy), ¬Green(x)) (On(mike, John),)
88 (On(x, yy), ¬Green(x)) (On(mike, John),)
89 (On(x, yy), ¬Green(x)) (On(mike, John),)
90 (On(x, yy), ¬Green(x)) (On(mike, John),)
91 (On(x, yy), ¬Green(x)) (On(mike, John),)
92 (On(x, yy), ¬Green(x)) (On(mike, John),)
93 (On(x, yy), ¬Green(x)) (On(mike, John),)
94 (On(x, yy), ¬Green(x)) (On(mike, John),)
95 (On(x, yy), ¬Green(x)) (On(mike, John),)
96 (On(x, yy), ¬Green(x)) (On(mike, John),)
97 (On(x, yy), ¬Green(x)) (On(mike, John),)
98 (On(x, yy), ¬Green(x)) (On(mike, John),)
99 (On(x, yy), ¬Green(x)) (On(mike, John),)
100 (On(x, yy), ¬Green(x)) (On(mike, John),)
```

可见该代码使用了相似的步骤最终正确的完成了一阶逻辑公式的推导。

2. 评测指标展示及分析

由于本程序使用的是暴力搜索的推导方法，因此代码的运行效率会相对较低，且很大程度上取决于子句的长度以及谓词内的项数等因素。

参考资料

《Fluent Python》 —— Luciano Ramalho 著