归结推理算法实验报告

武自厚 20336014 保密管理

2022 年 3 月 19 日

1 实验目的

根据 Robinson 归结推理的原理设计归结算法以实现自动推理.

2 算法原理

2.1 策略

本算法基于归结推理的“支持集策略”简化推理过程.“支持集策略”, 即仅有目标子句取反后新加入的子句 α 及其后代才能参与归结的策略.

2.2 自然语言描述

2.2.1 归结算法

对于给定前提集 F 以及命题 R, 将 F 转化为子句集 S_0,$\neg R$ 转化为子句 α 并添加到子句集中, 即

$$S := S_0 \cup \{\alpha\}$$

反复对 S 使用**单步归结算法**, 且选择的两个子句的规则满足支持集策略.

- 如果得到空子句 (), 则意味着 $S \vdash ()$, 也就是说 $F \models R$. 算法退出.
- 如果算法无法得到空子句, 则意味着 $F \not\models R$.

2.2.2 单步归结算法

对于两个子句, 使用**最一般合一算法**得到名称相同的原子及其否定. 然后删去它们, 再将两个子句合为一个新的子句, 添加到子句集 S 中.

2.2.3 最一般合一算法

对于两个谓词相同,“¬” 不同, 而参数不全都相同的文字. 如果两个文字不相同, 则寻找它们之间的一个差异项, 并将其中的一个变量替换为常量 (如果没有则是另一个变量), 替换加入最一般合一中. 循环往复直到两个文字的所有参数相同. 最后返回替换的集合作为最一般合一.

3 伪代码实现

3.1 主算法

Algorithm 1: 支持集策略归结推理算法

```
输入: 子句集 S, 目标子句 α
输出: S 是否能归结出 α 的布尔值 b
for s ∈ S do
    s 不在支持集中.
S := S ∪ {¬α}
while true do  // 无限循环
    S₀ := {}
    for x, y ∈ S do  // 遍历子句
        if x or y 在支持集中. then
            z := resolve(x, y)
            if z = () then
                return true
            else
                z 在支持集中.
                S₀ := S₀ ∪ {z}
    S := S ∪ S₀
```

3.2 单步归结算法

Algorithm 2: 单步归结算法 `resolve()`

输入: 两个子句 s, t

输出: 归结得出的新子句 u

$u := \{\}$

$\sigma := \{\}$

for 文字 $a \in s, b \in t$ **do**

  if a 与 b 的谓词相同, 且 "¬" 不同 **then**

    $\sigma := \sigma \cup$ `find_mgu`(a, b)

for 文字 $w \in s \cup t$ **do**

  $u := u \cup w\sigma$

return u

3.3 最一般合一算法

Algorithm 3: 最一般合一算法 `find_mgu()`

输入: 两个谓词相同的文字 w, v

输出: 两个文字的最一般合一替换 σ

$\sigma := \varepsilon$ 空代换

forall 两个文字中对应的参数 a, b **do**

  if $a \neq b$ **then**

    $t :=$ 其中的常量或变元

    $x :=$ 其中不是 t 或不在 t 中出现的变元

    if t, x 存在 **then**

      $\sigma := \sigma \circ \{t/x\}$

    else

      算法中止,MGU 不存在.

return σ

4 关键代码展示

4.1 支持集算法

```python
def reasoning(self):
    count = len(self.clauses) + 1
    resolved = []
    while True:
        buffer = []  # to store the new clauses because the
        ↪  resolutions are regard as synchronous.
        for this_clause in self.clauses:
            for that_clause in self.clauses:
                # traversal the cartesian product of the clauses
                unify_literal_pairs =
                ↪  this_clause.is_resolvable_at(that_clause)
                if this_clause != that_clause and
                ↪  len(unify_literal_pairs) != 0 and
                ↪  (self.is_prime[this_clause] or
                ↪  self.is_prime[that_clause]) and
                ↪  resolved.count((this_clause, that_clause)) == 0:
                    # The 1st is a new clause, the 2nd is MGU used to
                    ↪  make that clause
                    son_clause, mgu = this_clause.resolve(that_clause,
                    ↪  unify_literal_pairs)
                    places = place_to_str(self.get_places(this_clause,
                    ↪  that_clause, unify_literal_pairs))
                    buffer.append(son_clause)
                    print(str(count) + '. R' + places +
                    ↪  mgu_to_str(mgu) + ' = ' +
                    ↪  son_clause.__str__())
                    count += 1
                    resolved.append((this_clause, that_clause))
                    resolved.append((that_clause, this_clause))
                    if len(son_clause.literals) == 0:
                        return
        #
        for clause in buffer:
```

```
                self.is_prime[clause] = True
            self.extend(buffer)
```

同一代产生的子句应当是“同时”且“并发”生成的, 不能将先生成的子句直接加入子句集. 这里引入了一个列表作为缓冲 buffer, 将新的子句先装入 buffer, 等待通过原子句集能够归结出的所有子句生成后再将 buffer 中的子句一起并入子句集.

4.2 单步归结算法

```
        new_literals = []
        deduplicated_literals = []
        for literal in self.literals:
            if not include(unify_literals, literal):
                new_literals.append(copy.deepcopy(literal))

        for literal in other.literals:
            if not include(unify_literals, literal):
                new_literals.append(copy.deepcopy(literal))

        mgu = find_mgu(unify_literals)
        # replace the variables, following MGU.
        for i in range(len(new_literals)):
            new_literals[i] = replace(new_literals[i], mgu)

        # filter out the duplicated literals.
        for item in new_literals:
            for jtem in deduplicated_literals:
```

由于使用了 Python 中的列表, 可能产生重复元素, 需要在最后加入对重复元素的过滤.

4.3 最一般合一算法

```
def find_mgu(li: List[Tuple[Literal, Literal]]):
    mgu = {}
    for this, that in li:
        for this_arg, that_arg in zip(this.args, that.args):
```

```
            if is_variable(this_arg) and mgu.get(that_arg, that_arg) !=
            ↪  this_arg:
                mgu[this_arg] = that_arg
            elif is_variable(that_arg) and mgu.get(this_arg, this_arg) !=
            ↪  that_arg:
                mgu[that_arg] = this_arg
    return mgu
```

由于要求中并未出现函数, 所以不需要分析变量是否在常量中出现, 直接判定是否是变量即可. 合一替换以键值对数据结构表示.

5 实验结果及分析

5.1 测试样例

采用 "登山俱乐部" 问题为测试样例. 具体为:

$$
\begin{aligned}
K = \{ & \\
& A(\texttt{tony}), \\
& A(\texttt{mike}), \\
& A(\texttt{john}), \\
& L(\texttt{tony}, \texttt{rain}), \\
& L(\texttt{tony}, \texttt{snow}), \\
& (\neg A(x), S(x), C(x)), \\
& (\neg C(y), \neg L(y, \texttt{rain})), \\
& (L(z, \texttt{snow}), \neg S(z)), \\
& (\neg L(\texttt{tony}, u), \neg L(\texttt{mike}, u)), \\
& (L(\texttt{tony}, v), L(\texttt{mike}, v)), \\
& (\neg A(w), \neg C(w), S(w)), \\
\} &
\end{aligned}
$$

其中最后一个子句即是目标子句的否定 $\neg\alpha$.

5.2 实验结果

输出结果如下:

```
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[1a-11a]{w:=tony} = (¬C(tony), S(tony))
13. R[2a-11a]{w:=mike} = (¬C(mike), S(mike))
14. R[3a-11a]{w:=john} = (¬C(john), S(john))
15. R[6c-11b]{x:=w} = (¬A(w), S(w))
16. R[8b-11c]{z:=w} = (L(w, snow), ¬A(w), ¬C(w))
17. R[1a-15a]{w:=tony} = S(tony)
18. R[1a-16b]{w:=tony} = (L(tony, snow), ¬C(tony))
19. R[2a-15a]{w:=mike} = S(mike)
20. R[2a-16b]{w:=mike} = (L(mike, snow), ¬C(mike))
21. R[3a-15a]{w:=john} = S(john)
22. R[3a-16b]{w:=john} = (L(john, snow), ¬C(john))
23. R[6c-12a]{x:=tony} = (¬A(tony), S(tony))
24. R[6c-13a]{x:=mike} = (¬A(mike), S(mike))
25. R[6c-14a]{x:=john} = (¬A(john), S(john))
26. R[6c-16c]{x:=w} = (¬A(w), S(w), L(w, snow))
27. R[8b-12b]{z:=tony} = (L(tony, snow), ¬C(tony))
28. R[8b-13b]{z:=mike} = (L(mike, snow), ¬C(mike))
29. R[8b-14b]{z:=john} = (L(john, snow), ¬C(john))
30. R[8b-15b]{z:=w} = (L(w, snow), ¬A(w))
31. R[9a-16a, 9b-16a]{w:=mike, u:=snow} = (¬A(mike), ¬C(mike))
32. R[1a-23a] = S(tony)
33. R[1a-26a]{w:=tony} = (S(tony), L(tony, snow))
34. R[1a-30b]{w:=tony} = L(tony, snow)
35. R[2a-24a] = S(mike)
```

```
36. R[2a-26a]{w:=mike} = (S(mike), L(mike, snow))
37. R[2a-30b]{w:=mike} = L(mike, snow)
38. R[2a-31a] = ¬C(mike)
39. R[3a-25a] = S(john)
40. R[3a-26a]{w:=john} = (S(john), L(john, snow))
41. R[3a-30b]{w:=john} = L(john, snow)
42. R[6c-18b]{x:=tony} = (¬A(tony), S(tony), L(tony, snow))
43. R[6c-20b]{x:=mike} = (¬A(mike), S(mike), L(mike, snow))
44. R[6c-22b]{x:=john} = (¬A(john), S(john), L(john, snow))
45. R[6c-27b]{x:=tony} = (¬A(tony), S(tony), L(tony, snow))
46. R[6c-28b]{x:=mike} = (¬A(mike), S(mike), L(mike, snow))
47. R[6c-29b]{x:=john} = (¬A(john), S(john), L(john, snow))
48. R[6c-31b]{x:=mike} = (¬A(mike), S(mike))
49. R[8b-17a]{z:=tony} = L(tony, snow)
50. R[8b-19a]{z:=mike} = L(mike, snow)
51. R[8b-21a]{z:=john} = L(john, snow)
52. R[8b-23b]{z:=tony} = (L(tony, snow), ¬A(tony))
53. R[8b-24b]{z:=mike} = (L(mike, snow), ¬A(mike))
54. R[8b-25b]{z:=john} = (L(john, snow), ¬A(john))
55. R[8b-26b]{z:=w} = (L(w, snow), ¬A(w))
56. R[9a-18a]{u:=snow} = (¬L(mike, snow), ¬C(tony))
57. R[9b-20a]{u:=snow} = (¬L(tony, snow), ¬C(mike))
58. R[9a-26c, 9b-26c]{w:=mike, u:=snow} = (¬A(mike), S(mike))
59. R[9a-27a]{u:=snow} = (¬L(mike, snow), ¬C(tony))
60. R[9b-28a]{u:=snow} = (¬L(tony, snow), ¬C(mike))
61. R[9a-30a, 9b-30a]{w:=mike, u:=snow} = ¬A(mike)
62. R[1a-42a] = (S(tony), L(tony, snow))
63. R[1a-45a] = (S(tony), L(tony, snow))
64. R[1a-52b] = L(tony, snow)
65. R[1a-55b]{w:=tony} = L(tony, snow)
66. R[2a-43a] = (S(mike), L(mike, snow))
67. R[2a-46a] = (S(mike), L(mike, snow))
68. R[2a-48a] = S(mike)
69. R[2a-53b] = L(mike, snow)
70. R[2a-55b]{w:=mike} = L(mike, snow)
71. R[2a-58a] = S(mike)
```

```
72. R[2a-61a] = NIL
```

截图如下:

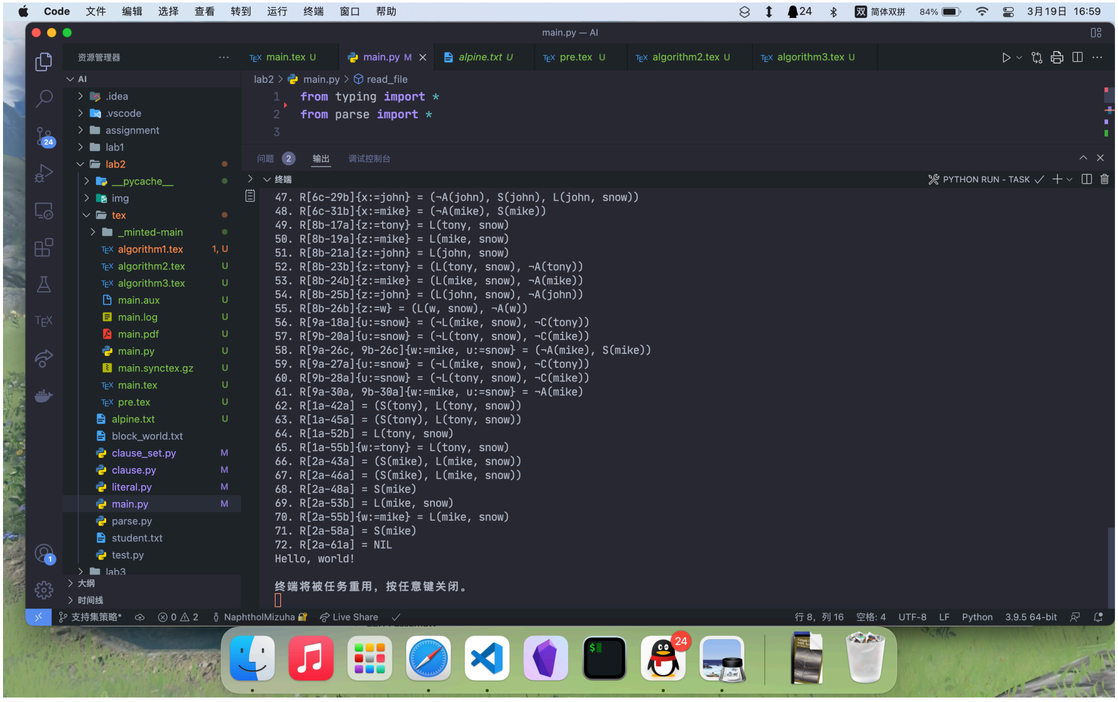

图 1: 结果展示

5.3 实验分析

该实验结果符合预期, 运行过程正常, 实验正常完成.

不过就结果展示而言, 输出了很多没有用到的子句. 若要改进, 可以记录最终空子句所有前驱子句, 并且只展示这些子句以达到简化输出, 使逻辑链条更为清晰, 便于用户理解.