

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



ĐỖ THANH NGHỊ

CƠ SỞ TRÍ TUỆ NHÂN TẠO

BÁO CÁO THỰC HÀNH LAB 2

Đỗ Thanh Nghị - 21120292

Thành phố Hồ Chí Minh – 2023

MỤC LỤC

MỤC LỤC	3
TỰ ĐÁNH GIÁ	4
PHẦN 1: GIẢI THÍCH MÃ NGUỒN	5
1.1. Những file code chính của mã nguồn :	5
1.2. Giải thích những hàm trong mã nguồn:.....	5
1.2.1. File <code>algorithm.py</code> :.....	5
1.2.2. File <code>main.py</code> :.....	12
PHẦN 2: THIẾT KẾ KỊCH BẢN	13
TÀI LIỆU THAM KHẢO	16

TỰ ĐÁNH GIÁ

STT	Đặc tả tiêu chí	Điểm	Tự đánh giá
1	Đọc dữ liệu đầu vào và lưu trong cấu trúc dữ liệu phù hợp	0.5	0.5
2	Cài đặt giải thuật hợp giải trên logic mệnh đề	1	1
3	Các bước suy diễn phát sinh đủ mệnh đề và kết luận đúng	2.5	2.5
4	Tuân thủ mô tả định dạng của đề bài	0.5	0.5
5	Báo cáo test case và đánh giá	0.5	0.5

PHẦN 1: GIẢI THÍCH MÃ NGUỒN

1.1. Những file code chính của mã nguồn :

- File main.py: chứa hàm main, chịu trách nhiệm chính cho việc đọc input, gọi những hàm xử lý bên file algorithm.py và xuất output theo đúng yêu cầu.
- File algorithm.py: chứa hàm chính PL_RESOLUTION như yêu cầu, ngoài ra còn chứa thêm những hàm phụ trợ cho hàm PL_RESOLUTION và các hàm để đọc Input và xuất Output, những hàm này sẽ được miêu tả cụ thể hơn ở phần 1.2.

1.2. Giải thích những hàm trong mã nguồn:

1.2.1. File algoritm.py:

- Hàm read_input: Tham số truyền vào của hàm chính là chuỗi đại diện cho đường dẫn đến file input cần đọc dữ liệu và output của hàm chính là list các Clause trong KB và string đại diện cho alpha.

```
1 def read_input(input_path: str) -> (list,str):
2     KB = []
3     alpha = ""
4     with open(input_path) as file:
5         alpha = file.readline()
6         # bỏ ký tự xuống dòng
7         alpha = alpha[:-1]
8         n = int(file.readline())
9
10        for i in range(n):
11            line = file.readline()
12            if (line[-1] == '\n'):
13                line = line[:-1]
14            literals = line.split(' OR ')
15            KB.append(tuple(literals))
16
17        return KB,alpha
```

- Hàm `write_output`: Tham số truyền vào chính là đường dẫn đến file output, list những Clause có sẵn từ KB và alpha và được sinh ra trong quá trình hợp giải, biến kết quả `res` có kiểu `bool` thể hiện rằng chúng ta có thể entail alpha từ KB không.

```

1  def write_output(output_path: str, info: list, res: bool):
2      with open(output_path, 'w') as file:
3          for item in info:
4              file.write(str(len(item)) + '\n')
5              for literals in item:
6                  outstr = ""
7                  if (len(literals) == 0):
8                      outstr = "{}"
9                  elif (len(literals) == 1):
10                     outstr += str(literals[0])
11                 else:
12                     for literal in literals[:-1]:
13                         outstr = outstr + literal + " OR "
14                     outstr += literals[-1]
15
16                 file.write(outstr + '\n')
17
18         if (res):
19             file.write("YES")
20         else: file.write("NO")

```

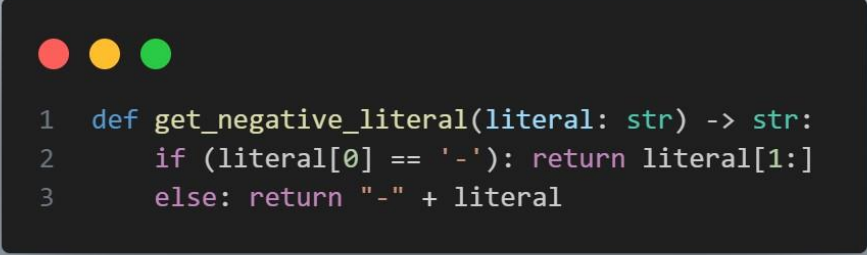
- Hàm `get_uni_ele_from_list`: Tham số truyền vào chính là 2 list và tham số trả về sẽ là một list. Hàm này có tác dụng là sẽ tìm ra những phần tử nào mà list 1 có mà list 2 không có để trả về. Hàm này được sử dụng để tìm ra những Clause mới được sinh ra sau một vòng lặp của hàm `PL_RESOLUTION`.

```

1  def get_uni_ele_from_a_list(l1: list, l2: list) -> list:
2      res = []
3      for i in l1:
4          if i not in l2:
5              res.append(i)
6      return res

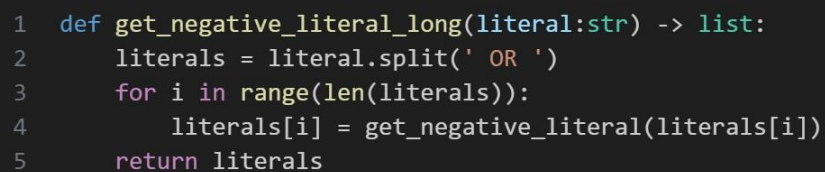
```

- Hàm `get_negative_literal`: Tham số truyền vào là một string đại diện cho một literal, tham số trả về sẽ là dạng phủ định của literal đó, ví dụ truyền vào “-A” thì sẽ trả về “A”.



```
1 def get_negative_literal(literal: str) -> str:
2     if (literal[0] == '-'): return literal[1:]
3     else: return "-" + literal
```

- Hàm `get_negative_literal_long`: Tham số truyền vào là một string và tham số trả về là một list. Hàm này có trách nhiệm chính trong chương trình chính là giúp chúng ta lấy được dạng phủ định của alpha chứa nhiều clause nối với nhau bằng phép OR (Ví dụ: “A OR B OR -C”) và trả về một list các clause để phục vụ cho quá trình hợp giải.



```
1 def get_negative_literal_long(literal: str) -> list:
2     literals = literal.split(' OR ')
3     for i in range(len(literals)):
4         literals[i] = get_negative_literal(literals[i])
5     return literals
```

- Hàm `get_literal_alpha`: Hàm này nhận vào một string chính là literal và trả về chính là chữ cái đầu tiên của literal đó.

```
1 def get_literal_alpha(literal: str) -> str:
2     if (literal[0] == '-'):
3         return literal[1]
4     else:
5         return literal[0]
```

- Hàm `compare_literal`: Tham số nhận vào chính là 2 string đại diện cho 2 literal, kết quả trả về sẽ là một giá trị int thể hiện cho kết quả so sánh. Nếu kết quả trả về là 1 thì literal 1 có thứ tự lớn hơn trong sắp xếp, giá trị trả về là 2 thì ngược lại và nếu giá trị trả về là 0 thì 2 literal có thứ tự bằng nhau trong việc sắp xếp. Hàm này giúp sắp xếp các clause theo 1 thứ tự nhất định để có thể quản lý những clause mới được sinh ra trong một vòng lặp của hàm `PL_RESOLUTION`.

```
1 # 0: bằng nhau
2 # 1: literal 1 lon hon
3 # 2: literal 2 lon hon
4 def compare_literal(l1:str, l2:str) -> int:
5     if (get_literal_alpha(l1) > get_literal_alpha(l2)):
6         return 1
7     elif (get_literal_alpha(l1) < get_literal_alpha(l2)):
8         return 2
9     else:
10        if (l1[0] == '-' and l2[0] != '-'):
11            return 2
12        elif (l1[0] != '-' and l2[0] == '-'):
13            return 1
14        else:
15            return 0
```

- Hàm PL_RESOLVE: Hàm này có tác dụng là hợp giải 2 mệnh đề lại với nhau. Tham số nhận vào sẽ là 2 mệnh đề (được lưu ở dạng list), và tham số trả ra sẽ là một list, list này chứa các clause mới được sinh ra trong quá trình hợp giải 2 mệnh đề đầu vào và các clause mới này được đảm bảo rằng không có các clause không có tác dụng cho việc suy luận như (clause V True) và được sắp xếp theo thứ tự alphabet.

```

1  def PL_RESOLVE(clause1: list, clause2: list) -> list:
2      resolvents = []
3
4      # để sử dụng hàm union ghép 2 mệnh đề này lại với nhau
5      # ta sẽ chuyển đổi 2 list từ input thành set
6      c1 = set(clause1)
7      c2 = set(clause2)
8
9      for i in c1:
10         if get_negative_literal(i) in c2:
11             new_clause = c1.union(c2) - {i, get_negative_literal(i)}
12
13             # kiểm tra xem có mệnh đề có dạng literal V True không
14             check = True
15
16             for literal in new_clause:
17                 if get_negative_literal(literal) in new_clause:
18                     check = False
19                     break
20
21             if(check):
22                 # sắp xếp các literal theo thứ tự ban đầu
23                 new_clause = list(new_clause)
24                 if '-' in new_clause:
25                     print(c1)
26                     print(c2)
27                     for i in range(len(new_clause)):
28                         for j in range(i+1, len(new_clause)):
29                             if compare_literal(new_clause[i], new_clause[j]) == 1:
30                                 new_clause[i], new_clause[j] = new_clause[j], new_clause[i]
31
32                     resolvents.append(tuple(new_clause))
33
34     return resolvents

```

Chi tiết cách hàm này hoạt động sẽ như sau:

- Giai đoạn chuẩn bị: Chúng ta sẽ tạo một list có tên là resolvents, list này có tác dụng lưu lại những mệnh đề mới được sinh ra

trong quá trình hợp giải. Sau đó chúng ta sẽ tạo ra 2 set từ 2 mệnh đề đầu vào, mục đích là để sử dụng các hàm có sẵn trong python như union để thuận hợp nhất 2 mệnh đề.

- Giai đoạn hợp giải: Chúng ta sẽ sử dụng một vòng for để duyệt lần lượt các literals có trong mệnh đề 1 (tạm gọi là c1) để xem thử có tồn tại literals đối nghịch trong mệnh đề 2 (tạm gọi là c2) hay không. Nếu có tồn tại thì ta sẽ bắt đầu kết hợp các phần tử ở 2 set là c1 và c2 (Ở c1 và c2 chắc chắn sẽ không có các literals trùng nhau ví dụ: A V A). Sau đó chúng ta sẽ duyệt qua mệnh đề mới được sinh ra xem có tồn tại 2 literals đối nghịch nhau hay không, nếu có thì không thêm vào resolvents, còn nếu không thì sắp xếp lại mệnh đề mới đó rồi thêm vào resolvents.
- Hàm PL_RESOLUTION: Hàm này chính là hàm chính của chương trình này có tác dụng xem xét rằng alpha có thể được entail từ knowledge base hay không. Tham số nhận vào của hàm này chính là list các mệnh đề trong KB và alpha. Hàm sẽ trả về một list lưu lại quá trình hợp giải mệnh đề và một biến bool trả về kết quả rằng ta có thể entail alpha từ knowledge base hay không.

```
1 def PL_RESOLUTION(KB: list,alpha: str) -> (bool,list):
2     if (len(alpha) <= 2):
3         alpha = get_negative_literal(alpha)
4         clause = KB + [(alpha,)]
5     else:
6         alpha = get_negative_literal_long(alpha)
7         for items in alpha:
8             KB += [(items,)]
9         clause = KB
10
11     new = set()
12
13     running_list = []
14
15     while 1:
16         n = len(clause)
17
18         for i in range(n):
19             for j in range(i+1,n):
20                 resolvent = PL_RESOLVE(clause[i],clause[j])
21                 new.update(resolvent)
22
23         new_ele = get_uni_ele_from_a_list(new,clause)
24
25         running_list.append(new_ele)
26
27         if(len(new_ele) == 0):
28             return False,running_list
29
30         if ({} in new):
31             return True,running_list
32
33         clause.extend(new)
```

Chi tiết cách hàm này hoạt động như sau:

- Giai đoạn chuẩn bị: ta sẽ chuyển alpha sang dạng phủ định và tạo một list mới có tên là clause, list này sẽ chứa cả KB và mệnh đề phủ định của alpha (-alpha). Ta sẽ tạo một set có tên là new để lưu lại những mệnh đề mới được sinh ra và đảm bảo chúng không trùng nhau. Tiếp theo đó ta sẽ tạo một list có tên là

running_list để lưu lại những mệnh đề mới được sinh ra theo từng vòng lặp.

- Giai đoạn suy luận: Ta sẽ dùng 1 vòng lặp vô tận while(1). Trong vòng lặp này chúng ta sẽ dùng hai vòng lặp for lồng nhau để duyệt qua tất cả các cặp mệnh đề trong clause. Đối với mỗi cặp mệnh đề ta sẽ gọi hàm PL_RESOLVE để tiến hành hợp giải và tiến hành cập nhật những mệnh đề mới vào new.

Sau khi kết thúc 2 vòng lặp for thì ta sẽ kiểm tra biến new. Hai điều kiện sau đây sẽ làm cho vòng lặp vô tận while(1) dừng lại:

- Trong new có một mệnh đề rỗng {} (1)
- New là tập con của clause, hay còn có nghĩa là không có mệnh đề mới nào được sinh ra. (2)
- Kết luận kết quả: Nếu ta rơi vào kết quả (1) thì nghĩa là alpha có thể được suy ra. Còn nếu ta rơi vào kết quả (2) thì nghĩa là alpha không thể được suy ra.

1.2.2. File main.py:

Hàm duy nhất của file này chính là hàm main. Hàm main có trách nhiệm tiến hành làm những công việc như sau:

Đi vào đường dẫn chứa những file input và lấy ra tất cả những file trong folder đó bằng hàm glob của thư viện glob.

Tiến hành đọc từng file input, lưu dữ liệu, tiến hành xử lý (gọi hàm PL_RESOLUTION và write_output).

```

1  def main():
2      # đường dẫn đến thư mục chứa script
3      script_path = os.path.abspath(__file__)
4      folder_path = os.path.dirname(script_path)
5
6      # lấy tất cả các input
7      input_files = glob.glob(folder_path + "/INPUT/*")
8
9      for i in range(len(input_files)):
10         input_files[i] = input_files[i].split("/INPUT/")[1]
11
12     for file in input_files:
13         KB, alpha = algorithm.read_input(folder_path + "/INPUT/" + file)
14         input_id = file.split("input")[1]
15         output_file = "output" + input_id
16         temp, out = algorithm.PL_RESOLUTION(KB, alpha)
17         algorithm.write_output(folder_path + "/OUTPUT/" + output_file, out, temp)

```

PHẦN 2: THIẾT KẾ KỊCH BẢN

Trong phần này, em sẽ trình bày 5 kịch bản mà em tự thiết kế cũng như output và cách hoạt động của thuật toán trên 5 kịch bản ấy.

Số hiệu x	input_x.txt	output_x.tx
0	B 4 A OR B OR -C -C A OR -B B OR C	5 A OR -C C A OR C B A OR B 2 A { YES
1	A 4 B OR D A OR C OR -B C OR -D A OR -C	6 -C A OR -B B OR C -B OR C A OR -D A OR C OR D 8 A OR D -B C A OR C -D B A OR B C OR D 3 D A { YES

2	D 5 C OR D -B OR -C OR -D -B OR C A OR D -A OR B	7 B OR D -A OR -C OR -D A OR -B OR -C C A -B OR -D -A OR C 8 -A OR -D A OR -B OR D A OR -B -C OR -D -A OR B OR -C -A OR -B OR -D B A OR -C OR D 6 -A OR B OR D B OR -C OR D B OR -C -B OR C OR D A OR -C -A OR -B OR C 1 -A OR C OR D 0 NO
3	A OR B OR -E 8 B OR C OR D A OR D OR E A OR B OR C OR F -B -F C OR D OR -F A OR E D	6 B OR C OR F D OR E A OR B OR C A OR B OR C OR D A OR C OR F C OR D 4 C OR F A OR C OR D A OR C B OR C 1 C 0 NO

4	-D OR E 7 A -A OR C B OR E -B OR -C -B OR -D D OR E OR F B OR D OR -F	10 -C OR E -C OR D OR -F -B D OR F C -A OR -B B -D OR E -B OR E OR F B OR D OR E 19 -A E OR F -B OR -C OR -F -C OR D -C -A OR D OR E -C OR D OR E -A OR D OR -F B OR D -B OR -C OR D OR E D OR E -D -C OR E OR -F D OR -F -B OR F B OR E OR -F E -A OR E {} YES
---	---	--

Nhận xét về thuật toán:

- Thuật toán luôn tìm ra được kết quả cuối cùng là có thể suy ra được alpha từ knowledge base hay không nếu dữ liệu đưa vào được chuẩn hóa đúng theo quy ước.
- Lượng mệnh đề mới sinh ra là khá nhiều nếu có nhiều literals và mệnh đề clauses.

TÀI LIỆU THAM KHẢO

Sách Artificial Intelligence: A Modern Approach, Third Edition, Chương 7, Hình 7.12, hàm PL-RESOLUTION