기초 인공지능(CSE4185) Assignment #3

2분반 20200183 박정원

1. 각 알고리즘마다 구현한 방법에 대한 설명

1) standardize_variables

```
standardized_rules = copy.deepcopy(nonstandard_rules)
variables = []
i = 0
for val in standardized_rules.values():
  isVarExist = False # proposition에 variable이 있으면 체크해준다.
  var = "x" + f"{i:0>4}" # 네 자리 정수 형태로 variable string을 생성
  for k, v in val.items():
    if k == "antecedents": # if key is antecedents
      if len(v) == 0: # meaning that this is triple
        break
      for lst in v:
        for j in range(len(lst)-1):
          if lst[j] == "something" or lst[j] == "someone":
            lst[j] = var # something or someone 을 이전에 생성한 variable로 바꾸
            isVarExist = True

    else: # consequent
      for j in range(len(v)-1):
        if v[j] == "something" or v[j] == "someone":
          v[j] = var
          isVarExist = True

  if isVarExist == True:
    i += 1
    variables.append(var)


return standardized_rules, variables
```

standardize_variables 함수에서 해야할 것은 rules가 input으로 들어오면, 각각의 rule 안에 antecedents와 consequent에 something 혹은 someone이 있으면, 고유한 variable로 바꿔주는 것이다. isVarExist 라는 변수를 통해서 하나의 rule 안에 변수로 바꾼 것이 있으면, i를 증가하여 다음 rule에서는 고유한 변수가 할당될 수 있도록 만들어주었다. 변수명은 x0000 와 같이 x 뒤에 네 자리 정수 형태가 되도록 문자열을 만들어주었다. something과 someone은 발견되는 즉시 변수로 바꾸고, 변수를 variables 리스트에 추가하는 방식으로 구현하였다.

2) unify

unify 함수 안에서 먼저 확인해야하는 것은 query와 datum이 unify 될 수 있는지 여부를 체크하는 것이다. query와 datum이 unify 되기 위해서는 eats나 is와 같은 predicate과 True와 False의 Boolean 값이 모두 동일해야 한다. 즉, variable

이 등장하는 인덱스를 제외한 나머지 모든 인덱스에서의 단어가 동일해야한다는 것이다.

```
for i in range(len(query)):
  if query[i] not in variables and datum[i] not in variables: # query와 datum 둘 다 variable이 아니면
    if query[i] != datum[i]: # 이 둘이 다르면, 무조건 unify 할 수 없다. (predicate이 다르거나, boolean 값이 다른 경우, etc)
      return None, None
```

query와 datum의 길이는 동일하기 때문에 i번째 인덱스의 단어가 query와 datum 둘 다 variable이 아니면, 해당 단어는 동일해야 하고, 만약 다르다면, 그 즉시 unify 함수는 None, None을 반환해야 한다.

```
while(True): # iterate until all the substitutions are done
  pair = None # if new unification is added, this pair variable has that unification pair. Otherwise, None
  for i in range(len(Q)):
    if Q[i] not in variables and D[i] not in variables: # No need to unify the not variable
      continue

    if Q[i] in variables: # query의 i번째 단어가 variable이면
      ret = unify_var(Q[i], D[i], subs) # unify_var 의 첫번째 인자로 Q[i]
    elif D[i] in variables: # datum의 i번째 단어가 variable이면
      ret = unify_var(D[i], Q[i], subs) # unify_var 의 첫번째 인자로 D[i]

    if ret != None: # unify_var 의 return 값이 None이 아니면, 새로운 unification이 만들어졌다는 뜻
      pair = ret # 해당 unification 을 pair에 대입
      # if new (key,val) is added, then all variable 'key' in query needs to be substituted to val first
      break # if new unifcation is added, before iterating to the end, we need to substitute the variable with the new set first

  if pair != None: # if new unification is returned
    key, val = pair[0], pair[1]
    for i in range(len(Q)):
      if Q[i] == key: # 새로운 unification에 맞게 variable을 unify 해준다.
        Q[i] = val
    subs[pair[0]] = pair[1] # add new 'x' : subs[x] to subs dictionary

  else: # nothing to be replaced, so done with substitution
    unification = Q
    return unification, subs # unification succeeds, no more unification to be added. so terminate.

return None, None # otherwise, unification fails
```

Q와 D는 각각 query와 datum을 deepcopy 한 것이다.

위에 예외처리를 통과했으면, unification이 더 이상 되지 않을 때까지 unify를 진행해야 하고, variable x와 해당 variable의 subs[x] pair를 subs 딕셔너리에 저장해야 한다.

결국 unify를 하다보면, 더 이상 새로운 unification이 만들어지지 않는 시점이 오게 될 것이다. (이미 subs 딕셔너리에 해당 variable의 key를 가진 값이 있다면) 이러한 작업을 하는 함수를 따로 선언해주었는데, 이는 바로 밑에서 설명할 unify_var 이라는 함수이다.

unify_var

```python
def unify_var(var, x, theta): # 새로운 unification을 찾는 함수
    '''
    inputs:
        var, a variable
        x, a variable or a constant
        theta, subs built so far in dictionary format
    '''
    if var in theta and theta[var] == x:
        return None
    elif x in theta and theta[x] == var:
        return None
    else:
        return (var, x) # new x and subs[x] to be added
```

unify_var의 입력으로는 var, x, theta를 받게 된다. 주석에도 설명을 해놨는데, var 은 무조건 variable이 들어오게 되고, x 변수에는 variable 혹은 constant 가 들어올 수 있다. theta는 현재까지 만들어진 subs 딕셔너리이다. 이제 var과 x에 대해서 각각이 이미 theta 딕셔너리에 있는 unification set이라면 None을 반환하도록 하고, 새롭게 추가될 수 있는 unification set이면 var, x 튜플을 반환한다.

```python
while(True): # iterate until all the substitutions are done
    pair = None # if new unification is added, this pair variable has that unification pair. Otherwise, None
    for i in range(len(Q)):
        if Q[i] not in variables and D[i] not in variables: # No need to unify the not variable
            continue

        if Q[i] in variables: # query의 i번째 단어가 variable이면
            ret = unify_var(Q[i], D[i], subs) # unify_var 의 첫번째 인자로 Q[i]
        elif D[i] in variables: # datum의 i번째 단어가 variable이면
            ret = unify_var(D[i], Q[i], subs) # unify_var 의 첫번째 인자로 D[i]

        if ret != None: # unify_var 의 return 값이 None이 아니면, 새로운 unification이 만들어졌다는 뜻
            pair = ret # 해당 unification 을 pair에 대입
            # if new (key,val) is added, then all variable 'key' in query needs to be substituted to val first
            break # if new unifcation is added, before iterating to the end, we need to substitute the variable with the new set first

    if pair != None: # if new unification is returned
        key, val = pair[0], pair[1]
        for i in range(len(Q)):
            if Q[i] == key: # 새로운 unification에 맞게 variable을 unify 해준다.
                Q[i] = val
        subs[pair[0]] = pair[1] # add new 'x' : subs[x] to subs dictionary

    else: # nothing to be replaced, so done with substitution
        unification = Q
        return unification, subs # unification succeeds, no more unification to be added. so terminate.

return None, None # otherwise, unification fails
```

다시 unify 함수로 돌아와서, while 문 안에서 체크를 해주는데, Q의 첫번째 인덱스부터 차례대로 돌면서 variable 인지 체크를 한다. Q[i]가 variable이라면, unify_var의 첫번째 인자로 Q[i]를 넘겨주고, 만약 Q[i]가 variable이 아니고, D[i]가 variable 이면, unify_var의 첫번째 인자로 D[i]를 넘겨준다. 이렇게 해서 unify 함수 내에서 요구한

```
Unification succeeds if (1) every variable x in the unified query is replaced by a variable or constant from datum, which we call subs[x], and (2) for any variable y in datum that matches to a constant in query, which we call subs[y], then every instance of y in the unified query should be replaced by subs[y].
```

(1) 번과 (2)번의 조건을 모두 만족시킬 수 있게 된다.

pair 변수는 None으로 초기화되어 있고, unify_var에서 반환한 값을 받는다. 만약 unify_var에서 반환한 것이 새로운 unification set이라면 그 즉시 break를 통해서 빠져나온다. 그 이유는 새로운 unification set을 발견하면 해당 variable은 모두 먼저 치환을 진행한 후에 새로운 variable의 unification을 찾아야 하기 때문이다.

Pair가 None이 아니라면, 새로운 unification 이 추가될 수 있다는 뜻이기 때문에 명제에서 해당 variable을 모두 치환해주고, subs 딕셔너리에 pair를 추가해준다.

Pair가 None이면, 더 이상 새로운 Unification이 나타나지 않고, unified 되었다는 뜻이기 때문에 unification 변수에 unified 된 query Q를 할당하고, subs와 함께 튜플 형태로 return 해준다.

3) apply

```python
R = copy.deepcopy(rule)
G = copy.deepcopy(goals)
applications = []
goalsets = []

for i, pro in enumerate(G): # fetch each proposition from goals
  uni, subs = unify(R["consequent"], pro, variables) # check if unification is possible for each goal with rule's consequent
  if uni == None and subs == None: # if unification impossible
    continue
  new_dict = {}
  new_dict["antecedents"] = []
  new_dict["consequent"] = uni # add unified consequent to applications

  new_goal = copy.deepcopy(G)
  del new_goal[i] # delete the goal that unified with applications[i]["consequent"]

  for ant in R["antecedents"]: # add unified antecedents to applications
    if bool(subs) == False: # if the subs dictionary is empty, it means that there was no variable in the consequent.
      new_dict["antecedents"].append(ant) # add an unifed antecedent to the dictionary
      new_goal.append(ant) # add an unified antecedent to the new_goal
    else:
      unified_ant = substitute(ant, subs, variables) # unify the antecedents
      new_dict["antecedents"].append(unified_ant)
      new_goal.append(unified_ant) # add unified antecedents to new_goal

  applications.append(new_dict)
  goalsets.append(new_goal)

return applications, goalsets
```

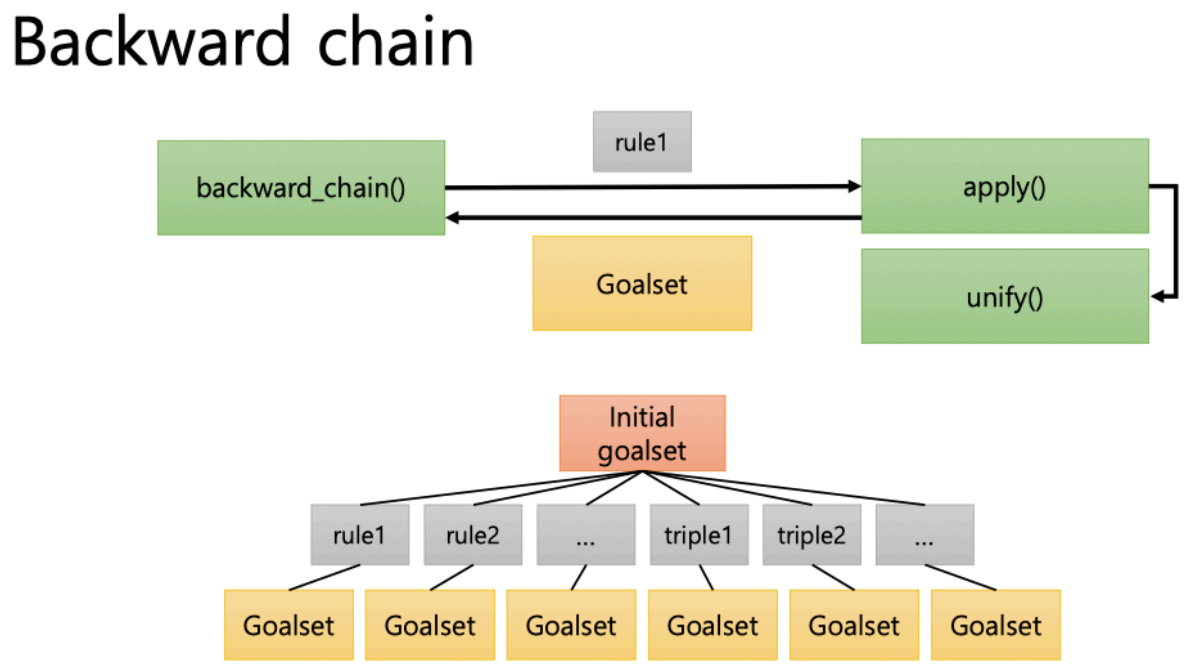
apply 함수는 rule과 goal이 주어지면, 해당 goal과 rule의 consequent가 unify 될 수 있는지 판단하고, unify 될 수 있다면, consequent를 unify 하고, antecedents도 unify 한 후에 새로운 newgoals를 반환하는 함수이다. 즉, backward_chain 에서 최종 goal 에서부터 역으로 새로운 goal을 도출해나가야 하는데, 그런 역할을 하는 함수가 바로 apply 함수인 것이다.

위 내용을 구현한 것이 위에 스크린샷으로 첨부한 코드이다. G에는 입력으로 주어진 goals이, R에는 rule이 들어가 있다. 각각의 goal를 for 문으로 돌면서, 각각의 goal에 대해서 입력으로 주어진 rule의 consequent와 unify 될 수 있는지 판단하고, 만약 uni와 subs가 모두 None, None이면 unify 될 수 없다는 뜻이므로, continue를 시켜준다. 우리가 최종적으로 반환할 applications 딕셔너리에는 각각의 goal에 대해 unify 된 antecedents와 consequent가 들어가야 한다. 즉, 해당 모양에 맞게 초기화를 해준다. 우리가 반환할 goalsets에는 unify 된 rule의 consequent를 제외하고 해당 rule의 antecedents로 대체된 goals를 반환해야 한다. 그래야 새로운 goal이 또 다른 query가 될 것이기 때문이다. 각각의 antecedents에 대해서 for 문을 돌면서, substitute 함수를 통해서 variable을 모두 치환해준다. 그리고 치환된 antecedent를 각각 application에 추가될 딕셔너리의 antecedent에 추가하고, goalsets에 추가될 리스트인 new_goal에도 추가해준다. 이 때, subs에 아무런 key가 없는 경우에는 antecedent가 치환될 수 없는 상태이기 때문에 해당 antecedent를 그대로 추가한다.

```
def substitute(pro, subs, variables): # 해당 명제 안에 있는 variable을 현재의 unification set에 있는 값으로 교체
  proposition = copy.deepcopy(pro)
  for i, entry in enumerate(proposition):
    if entry in variables:
      if entry in subs:
        proposition[i] = subs[entry]
  return proposition # return new substituted proposition
```

substitute 함수는 antecedent의 variable을 치환하기 위해서 따로 선언한 함수이다. apply에서 rule의 consequent를 unify 할 때 받아온 subs 딕셔너리의 key와 value 쌍으로 antecedent에도 해당 variable를 치환해주고, 치환된 명제를 반환해주면 된다.

4) **backward_chain**



조교님의 발표자료에서 가져온 backward chain의 구조입니다.

먼저 backward chain 함수의 인자로는 query, rules, variables 가 들어온다. query 로 들어오는, 즉 증명해야 하는 question이 있는데, 이 Initial goalset으로부터 rules를 탐색해보면서, apply를 하고 새로 받아온 goalset이 다시 goal이 되어서 재귀적으로 증명을 하는 식으로 구현을 하면 되므로, 사실상 미리 구현해둔 apply()를 활용하면 된다.

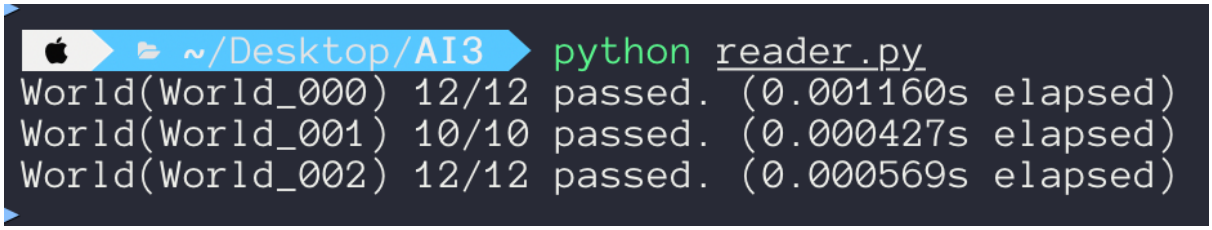
```
Q = []
Q.append(copy.deepcopy(query))
T = {}
for key, r in rules.items(): # preprocess the triples first
  if key[0] == 't': # if key is the triple
    if r["consequent"][:3] == Q[0][:3]: # if the query before the boolean is equal to triple's consequent
      if r["consequent"][3] != Q[0][3]: # if the boolean is different
        return None # if all the words other than boolean value is same, but only the boolean is different, then it means the statement is false
      else:
        return True # if two statements are same, immediately return True
    continue
  else: # if key is the rule
    if r["consequent"] == Q[0]: # if rule's consequent is exactly same with the query
      return True


for key, r in rules.items():
  if key[0] == 't': # meaning that the key is triple
    continue

  if r["consequent"] == Q[0]: # if rule's consequent is exactly same with the query
    continue

  applications, goalsets = apply(r, Q, variables)
  if len(goalsets) == 0: # meaning that this rule has nothing to do with the current goal
    continue
  succeed = []
  for goal in goalsets:
    for g in goal:
      succeed.append(backward_chain(g, rules, variables)) # all the antecedents should be True in order to make the consequent true

  true_cnt = 0
  for i in range(len(succeed)): # count the number of Trues
    if succeed[i] == True:
      true_cnt += 1

  if true_cnt == len(succeed): # if the length of the list is equal to the number of Trues, it means the query is also true
    return True

return None
```

해당 rules를 for 문에서 돌면서 만약 triple이라면 현재 증명해야 하는 query와 같은지 판단을 한다. 만약 boolean 이전의 단어들이 모두 같은데, boolean 값이 다른 경우에는 해당 명제가 틀렸다는 뜻이므로, None을 반환해야 한다. 만약 모두 같으면, True를 반환한다. 이렇게 먼저 triple을 모두 확인해서 우리가 증명해야 하는 명제가 True 인지 None 인지 바로 판단할 수 있는지 체크한다. rule 인 경우에도 consequent에 variable이 없고, query와 동일할 수 있다. 이런 경우에는 antecedents의 참 거짓 유무와 상관없이 미리 True를 반환해준다.

이제 나머지 rules 들을 iterate 하는데, 먼저 apply 함수를 통해서 새로운 goalsets를 받아온다. 새로운 goalsets에서 각각의 goal들이 모두 True 여야 우리가 증명해야 하는 query가 True가 되기 때문에 succeed 리스트를 만들어서 goalsets의 각각의 goal에 대해서 backward_chain을 해서 return 값을 succeed 리스트에 추가해준다. 그리고 succeed 리스트에서 True의 개수를 세어주고, succeed 리스트의 길이와 개수가 같으면, 그제서야 우리가 증명해야 하는 명제의 True를 반환해준다.

이 모두가 아닌 경우에는 None을 반환하도록 한다.

2. 실행 캡처 화면


~/Desktop/AI3
python reader.py
World(World_000) 12/12 passed. (0.001160s elapsed)
World(World_001) 10/10 passed. (0.000427s elapsed)
World(World_002) 12/12 passed. (0.000569s elapsed)