In [1]:
```
import numpy as np
```

Case I: Packer

facts

In [2]:
```
observation={}
#suitcase
observation["suitcase"]=[]
observation["suitcase size"]=np.random.randint(30,50)
#mound
observation["mound"]=["shirt","shirt","suit"]
#essentials
observation["essentials"]=["toiletry"]

#articles
observation['articles']=["alarm clock","belt","coat","clothes brush","dressing gown",\
                         "handkerchief","pajamas","reading material","shirt","shoe brush"\
                         ,"shoe polish","shoes","shorts","slippers","socks","suit","sweater"\
                         ,"tie","toiletry","towel","trousers","underclothes","writing material"]
observation['article size']={}
for item in observation['articles']:
    observation['article size'][item]=np.random.randint(1,10)

#mate
observation['mate']={}
observation['mate']["shoe brush"]=["shoe polish","shoes"]
observation['mate']["shoe polish"]=["shoe brush","shoes"]
observation['mate']["shoes"]=["shoe polish","shoe brush"]
observation['mate']["suit"]=["tie"]
observation['mate']["tie"]=["suit"]
observation['mate']["reading material"]=["writing material"]
observation['mate']["writing material"]=["reading material"]

#step
observation["step"]="checking"
observation["step_status"]="keep"
observation['finished']=False
```

rule base

In [3]:
```
def PACKER(observation):
    observation_t=str(observation)
    if observation["step"]=="checking":
        for ess_item in observation["essentials"]:
            if ess_item not in observation["mound"]:
                print("adding {} in to the mound".format(ess_item))
                observation["mound"].append(ess_item)
        if str(observation)!=str(observation_t):
            return observation

    if observation["step"]=="checking":
        if len(observation["mound"])>0:
            for item in observation["mound"]:
                if item in observation["mate"]:
                    for mate in observation["mate"][item]:
                        if mate not in observation["mound"]:
                            print("adding {} in to the mound since it is the mate of {}".format(mate,item))
                            observation["mound"].append(mate)
        observation["step_status"]="abandon"
        if str(observation)!=str(observation_t):
            return observation

    if observation["step"]=="checking":
        if observation["step_status"]=="abandon":
            print("start to put articles to suitcase")
            observation["step"]="putting"
        if str(observation)!=str(observation_t):
            return observation

    if observation["step"]=="putting":
        if len(observation["mound"])>0:
            for item in observation["mound"]:
                if observation['article size'][item]<=observation["suitcase size"]:
                    observation["suitcase"].append(item)
                    observation["suitcase size"]-=observation['article size'][item]
                    observation['mound'].remove(item)
                    print("adding {} in to the suitcase".format(item))
        if str(observation)!=str(observation_t):
            return observation

    if observation["step"]=="putting":
        if len(observation["mound"])>0:
            for item in observation["mound"]:
                if observation['article size'][item]>observation["suitcase size"]:
                    observation['finished']=True
                    print("cannot add {} in to the suitcase".format(item))
                    break
            print("packing is finished".format(item))
        if str(observation)!=str(observation_t):
            return observation

    if observation["step"]=="putting":
        if len(observation["mound"])==0:
            observation['finished']=True
            print("all articles in mound are added")
            print("packing is finished")
        return observation
```

Inference engine

In [4]:
```
while not observation['finished']:
    observation=PACKER(observation)
    print("###facts are updated###")
observation['suitcase']
```

```
adding toiletry in to the mound
###facts are updated###
adding tie in to the mound since it is the mate of suit
###facts are updated###
start to put articles to suitcase
###facts are updated###
adding shirt in to the suitcase
adding suit in to the suitcase
adding tie in to the suitcase
###facts are updated###
adding shirt in to the suitcase
###facts are updated###
adding toiletry in to the suitcase
###facts are updated###
all articles in mound are added
packing is finished
###facts are updated###
```

Out[4]:
```
['shirt', 'suit', 'tie', 'shirt', 'toiletry']
```

Case II: Identifier

Facts

In [5]:
```
final_hypothesis_item=["undergraduate","graduate","secretary","professor","dean","visitor"]

rule_elements=['undergraduate','graduate','secretary','professor','dean',\
               'visitor','student','academic','administrator','emaciated',\
               'wears a helmet','eats junk food','talks to himself','takes long lunches',\
                'writes memos','wears a suit','shuffles papers','looks confused','looks sleepy']

facts={}
for element in rule_elements:
    facts[element]=None

facts['emaciated']=True
facts['wears a helmet']=True
facts['eats junk food']=True
facts['looks confused']=True
facts['visitor']=True
```

Rule base

In [6]:
```
Rulebase={}
for element in rule_elements:
    Rulebase[element]=[None]

Rulebase["undergraduate"]=["student","emaciated","wears a helmet","AND"]
Rulebase["graduate"]=["student","emaciated","AND"]
Rulebase["student"]=["academic","eats junk food","AND"]
Rulebase["secretary"]=["administrator","talks to himself","AND"]
Rulebase["professor"]=["academic","talks to himself","takes long lunches","AND"]
Rulebase["dean"]=["administrator","writes memos","takes long lunches","AND"]
Rulebase["administrator"]=["wears a suit","shuffles papers","AND"]
Rulebase["academic"]=["looks confused","looks sleepy","OR"]
```

Inference engine (Forward Chaining)

In [7]:
```
finished=False
while not finished:
    for element in facts:
        if facts[element]!=True and not finished:
            proof_set=[]
            for key in Rulebase[element][:-1]:
                proof_set.append(facts[key])
            if Rulebase[element][-1]=="AND":
                if len(set(proof_set))==1 and proof_set[0]:
                    print("{} ({})=> {}".format(Rulebase[element][:-1],Rulebase[element][-1],element))
                    facts[element]=True
                    if element in final_hypothesis_item:finished=True
            else:
                if True in proof_set:
                    print("{} ({})=> {}".format(Rulebase[element][:-1],Rulebase[element][-1],element))
                    facts[element]=True
                    if element in final_hypothesis_item:finished=True
```

```
['looks confused', 'looks sleepy'] (OR)=> academic
['academic', 'eats junk food'] (AND)=> student
['student', 'emaciated', 'wears a helmet'] (AND)=> undergraduate
```

Inference engine (Backward Chaining)

In [8]:
```
facts={}
for element in rule_elements:
    facts[element]=None

facts['emaciated']=True
facts['wears a helmet']=True
facts['eats junk food']=True
facts['looks confused']=True
facts['visitor']=True
Rulebase={}
for element in rule_elements:
    Rulebase[element]=[None]

Rulebase["undergraduate"]=["student","emaciated","wears a helmet","AND"]
Rulebase["graduate"]=["student","emaciated","AND"]
Rulebase["student"]=["academic","eats junk food","AND"]
Rulebase["secretary"]=["administrator","talks to himself","AND"]
Rulebase["professor"]=["academic","talks to himself","takes long lunches","AND"]
Rulebase["dean"]=["administrator","writes memos","takes long lunches","AND"]
Rulebase["administrator"]=["wears a suit","shuffles papers","AND"]
Rulebase["academic"]=["looks confused","looks sleepy","OR"]
```

In [9]:
```
def check_evidence(element,facts,Rulebase):
    print("check facts for '{}'".format(element))
    truth=None
    if len(Rulebase[element][:-1])>0:
        print("need to find evidence for",Rulebase[element][:-1])

    if facts[element]!=True:
        proof_set=[]
        for key in Rulebase[element][:-1]:
            truth,facts,Rulebase=check_evidence(key,facts,Rulebase)
            proof_set.append(facts[key])
        if Rulebase[element][-1]=="AND":
            if len(set(proof_set))==1 and proof_set[0]:
                print("{} ({})=> {}".format(Rulebase[element][:-1],Rulebase[element][-1],element))
                print("***{} is proved***".format(element))
                facts[element]=True
                if element in final_hypothesis_item:truth=True
        elif Rulebase[element][-1]=="OR":
            if True in proof_set:
                print("{} ({})=> {}".format(Rulebase[element][:-1],Rulebase[element][-1],element))
                print("***{} is proved***".format(element))
                facts[element]=True
                if element in final_hypothesis_item:truth=True
    else:
        print("It is:",facts[element])
        truth=True
    return truth,facts,Rulebase
```

In [10]:
```
finished=False
goal="undergraduate"
truth,facts,Rulebase=check_evidence(goal,facts,Rulebase)
if facts[goal]==True:
    finished=True
else:
    print("***failed to prove the hypothesis***")
```

```
check facts for 'undergraduate'
need to find evidence for ['student', 'emaciated', 'wears a helmet']
check facts for 'student'
need to find evidence for ['academic', 'eats junk food']
check facts for 'academic'
need to find evidence for ['looks confused', 'looks sleepy']
check facts for 'looks confused'
It is: True
check facts for 'looks sleepy'
['looks confused', 'looks sleepy'] (OR)=> academic
***academic is proved***
check facts for 'eats junk food'
It is: True
['academic', 'eats junk food'] (AND)=> student
***student is proved***
check facts for 'emaciated'
It is: True
check facts for 'wears a helmet'
It is: True
['student', 'emaciated', 'wears a helmet'] (AND)=> undergraduate
***undergraduate is proved***
```

Package PyKnow

see https://pyknow.readthedocs.io/en/stable/introduction.html (https://pyknow.readthedocs.io/en/stable/introduction.html)

In [11]:
```
import sys
sys.path.append("./family_relations/")
import driver
```

In [12]:
```
driver.fc_test('jamie')
```

```
doing proof
jamie, jim are ('daughter', 'father')
jamie, sandy_w are ('daughter', 'mother')
jamie, bill are (('grand', 'daughter'), ('grand', 'father'))
jamie, elvina are (('grand', 'daughter'), ('grand', 'mother'))
jamie, allen are (('great', 'grand', 'daughter'), ('great', 'grand', 'father'))
jamie, ismay are (('great', 'grand', 'daughter'), ('great', 'grand', 'mother'))
jamie, jimjim are ('sister', 'brother')
jamie, johnjohn are ('sister', 'brother')
jamie, steve_w are ('niece', 'uncle')
jamie, jeri are ('niece', 'aunt')
jamie, annette are ('niece', 'aunt')
jamie, helen_w are ('niece', 'aunt')
jamie, mary_w are ('niece', 'aunt')
jamie, john_w are (('great', 'niece'), ('great', 'uncle'))
jamie, chuck_w are (('great', 'niece'), ('great', 'uncle'))
jamie, norma are (('great', 'niece'), ('great', 'aunt'))
jamie, david_w are ('1st', 'cousins')
jamie, jessica are ('1st', 'cousins')
jamie, bridget are ('1st', 'cousins')
jamie, brian2 are ('1st', 'cousins')
jamie, victoria are ('1st', 'cousins')
jamie, charli are ('2nd', 'cousins')
jamie, m_thomas are ('2nd', 'cousins')
jamie, david_a are ('2nd', 'cousins')
jamie, mitch are ('1st', 'cousins', 1, 'removed')
jamie, jonni are ('1st', 'cousins', 1, 'removed')
jamie, lorri are ('1st', 'cousins', 1, 'removed')
jamie, bruce are ('1st', 'cousins', 1, 'removed')
jamie, fred_a are ('1st', 'cousins', 1, 'removed')
jamie, tim are ('1st', 'cousins', 1, 'removed')
jamie, vicki are ('1st', 'cousins', 1, 'removed')
jamie, jill are ('1st', 'cousins', 1, 'removed')

done
family: 9 fact names, 94 universal facts, 6920 case_specific facts
fc_example: 20 fc_rules, 6772 triggered, 892 rerun
fc_example: 0 bc_rules, 0 goals, 0 rules matched
            0 successes, 0 failures
fc time 0.47, 14628 asserts/sec
```

In [13]:
```
driver.general(person1='bruce',relationship=('father', 'son'))
```

```
doing proof
bruce, m_thomas are ('father', 'son')
bruce, david_a are ('father', 'son')

done
bc2_example: 0 fc_rules, 0 triggered, 0 rerun
bc2_example: 29 bc_rules, 105 goals, 390 rules matched
             82 successes, 390 failures
family: 9 fact names, 94 universal facts, 0 case_specific facts
bc time 0.01, 7535 goals/sec
```

In [14]:
```
driver.test('jamie')
```

```
doing proof
jamie, jim are daughter, father
jamie, sandy_w are daughter, mother
jamie, bill are grand daughter, grand father
jamie, elvina are grand daughter, grand mother
jamie, allen are great grand daughter, great grand father
jamie, ismay are great grand daughter, great grand mother
jamie, jimjim are sister, brother
jamie, johnjohn are sister, brother
jamie, steve_w are niece, uncle
jamie, jeri are niece, aunt
jamie, annette are niece, aunt
jamie, helen_w are niece, aunt
jamie, mary_w are niece, aunt
jamie, john_w are great niece, great uncle
jamie, chuck_w are great niece, great uncle
jamie, norma are great niece, great aunt
jamie, john_w are great niece, great uncle
jamie, chuck_w are great niece, great uncle
jamie, norma are great niece, great aunt
jamie, david_w are 1st cousins
jamie, jessica are 1st cousins
jamie, bridget are 1st cousins
jamie, brian2 are 1st cousins
jamie, victoria are 1st cousins
jamie, charli are 2nd cousins
jamie, m_thomas are 2nd cousins
jamie, david_a are 2nd cousins
jamie, mitch are 1st cousins, 1 removed
jamie, jonni are 1st cousins, 1 removed
jamie, lorri are 1st cousins, 1 removed
jamie, bruce are 1st cousins, 1 removed
jamie, fred_a are 1st cousins, 1 removed
jamie, tim are 1st cousins, 1 removed
jamie, vicki are 1st cousins, 1 removed
jamie, jill are 1st cousins, 1 removed

done
example: 6 fc_rules, 262 triggered, 0 rerun
example: 21 bc_rules, 7375 goals, 15317 rules matched
         1560 successes, 15317 failures
family: 9 fact names, 94 universal facts, 422 case_specific facts
fc time 0.01, 47118 asserts/sec
bc time 0.84, 8745 goals/sec
total time 0.85
```