9 January 2022

Artificial Intelligence

Laboratory activity

Name:Cioara Iulia Maria
Group:30231
Email:cioara_iulia@yahoo.com

Teaching Assistant: Adrian Groza
Adrian.Groza@cs.utcluj.ro

Contents

Chapter 1

A1: Search

Implementarea algoritmilor de cautare in cazul Pacman, urmeaza sa ii detaliez in continuare pe fiecare in parte.

1. (Q1) Depth-first search:

   Acest algoritm pleaca dintr-un nod radacina si gaseste toti succesorii acestuia urmand sa aleaga unul dintre ei. Am folosit o structura pentru a salva nodurile deja vizitate si o lista pentru a adauga nodurile.

```
    def depthFirstSearch(problem):
        visited = []
        start_node = problem.getStartState()
        if problem.isGoalState(start_node):
            return []
        my_stack = util.Stack()
        my_stack.push((start_node, []))
        while not my_stack.isEmpty():
      curr_node, actions = my_stack.pop()
      if curr_node not in visited:
        visited.append(curr_node)
        if problem.isGoalState(curr_node):
          return actions
          for succesor,next_action,
          cost in problem.getSuccessors(curr_node):
            new_action = actions + [next_action]
       my_stack.push((succesor, new_action))

```

2. (Q2) Breadth-first search:

   Acest algoritm porneste dintr-un nod radacina si viziteaza prima data nodurile vecine acestuia inainte de a trece la nivelul urmator. Am folosit o structura pentru a salva nodurile deja vizitate si o coada pentru a adauga nodurile.

```
    def breadthFirstSearch(problem):
             visited = []
        start_node = problem.getStartState()
         if problem.isGoalState(start_node):
            return []
                 my_queue = util.Queue()
                 my_queue.push((start_node, []))
                 while not my_queue.isEmpty():
      curr_node, actions = my_queue.pop()
      if curr_node not in visited:
       visited.append(curr_node)
       if problem.isGoalState(curr_node):
```

```
        return actions
        for succesor, next_action, cost in problem.getSuccessors(
curr_node):
            new_action = actions + [next_action]
            my_queue.push((succesor, new_action))

```

3. (Q3) Uniform-cost search:

   Acest algoritm este similar cu BFS si DFS doar ca foloseste cel mai mic cost cumulat pentru a gasi un drum de la sursa la destinatie. Ca structura am folosit o coada de prioritati.

```
def uniformCostSearch(problem):
    visited = []
    start_node = problem.getStartState()
    if problem.isGoalState(start_node):
        return []
    my_priorityQ = util.PriorityQueue()
    my_priorityQ.push((start_node, [], 0), 0)
    while not my_priorityQ.isEmpty():
        curr_node, actions, cost = my_priorityQ.pop()
        if curr_node not in visited:
            visited.append(curr_node)
            if problem.isGoalState(curr_node):
                return actions
            for successor, next_action, next_cost in problem.
getSuccessors(curr_node):
                new_action = actions + [next_action]
                priority = cost + next_cost
                my_priorityQ.push((successor, new_action, priority),
priority)

```

4. (Q4) A* search:

   Acest algoritm este similar cu uniform cost search, diferenta fiind ca acesta isi calculeaza si costul pentru a ajunge la tinta. Functia pe care o foloseste este f(n)=g(n)+h(n).

```
def aStarSearch(problem, heuristic=nullHeuristic):
    visited = []
    start_node = problem.getStartState()
    if problem.isGoalState(start_node):
        return []
    my_priorityQ = util.PriorityQueue()
    my_priorityQ.push((start_node, [], 0), 0)
    while not my_priorityQ.isEmpty():
        curr_node, actions, p_cost = my_priorityQ.pop()
        if curr_node not in visited:
            visited.append(curr_node)
            if problem.isGoalState(curr_node):
                return actions
            for next, action, cost in problem.getSuccessors(curr_node):
                new_action = actions + [action]
                new_cost = p_cost + cost
                new_heuristic = new_cost + heuristic(next, problem)
                my_priorityQ.push((next, new_action, new_cost),
new_heuristic)

```

5. (Q5) Greedy:

   Acest algoritm este o combinatie intre BFS si A* doar ca foloseste o functie heuristica diferita. Calculeaza la nivel local cea mai optima alegere pentru a gasi un optim global.

```
def greedy(problem, heuristic=nullHeuristic):
  start_node = problem.getStartState()
  visited = []
  if problem.isGoalState(start_node):
      return []
  my_priorityQ = util.PriorityQueue()
  my_priorityQ.push((start_node, [], 0), 0)
  while not my_priorityQ.isEmpty():
      curr_node, actions, p_cost = my_priorityQ.pop()
      if curr_node not in visited:
          visited.append(curr_node)
          if problem.isGoalState(curr_node):
              return actions
          for next, action, cost in problem.getSuccessors(curr_node):
              new_action = actions + [action]
              new_cost = p_cost + cost
              my_priorityQ.push((next, new_action, new_cost),
 heuristic(next, problem))

```

6. (Q6) Iterative Deepening Search:

   Acest algoritm rezuma procesul de cautare intr-o singura functie prin executarea consecutiva a primelor cautari in adancime la niveluri de adancime din ce in ce mai mari, marcate ca si deep. Aici verificam adancimea pe care am atins-o in succesorii nodului curent intr-o bucla.

```
def iterativeDeepeningSearch(problem):
  start_node = problem.getStartState()
  my_stack = util.Stack()
  my_stack.push((start_node, [], 0))
  deep= 0
  while not my_stack.isEmpty():
      deep += 1
      curr_node, actions, cost = my_stack.pop()
      visited = []
      visited.append(curr_node)
      while True:
          for succesor, next_action, next_cost in problem.
 getSuccessors(curr_node):
              if succesor not in visited and (cost + next_cost) <=
 deep:
                  visited.append(succesor)
                  my_stack.push((succesor, actions + [next_action],
 cost + next_cost))
          if my_stack.isEmpty():
              break
          curr_node, actions, cost = my_stack.pop()
          if problem.isGoalState(curr_node):
              return actions
      my_stack.push((start_node, [], 0))

```

Chapter 2

A2: Logics

Pentru aceasta parte am ales sa fac cateva exemple de logic puzzle.

1. Knights and Knaves and Spies

Pe o insula se afla cavaleri, valeti si spioni. Esti abordat de 3 oameni care poarta haine de culori diferite. Stii ca unul este cavaler, unul este valet si unul este spion. Ei vorbesc in urmatoarea ordine:

(a) Barbatul care poarta albastru spune: "Sunt cavaler"

(b) Barbatul care poarta rosu spune: "El spune adevarul"

(c) Barbatul care poarta rosu spune: "Sunt spion"

Cine este cavaler,valet si spion?

Codul pentru informatiile care reies din textul dat:

```
formulas(assumptions).

cavaler(x)-> -valet(x) & -spion(x).
valet(x) -> -cavaler(x) & -spion(x).
spion(x) -> -cavaler(x) & -valet(x).
```

```
-cavaler(x) & -valet(x) -> spion(x).

cavaler(A) | valet(A) | spion(A).
cavaler(B) | valet(B) | spion(B).
cavaler(C) | valet(C) | spion(C).

cavaler(A) -> -cavaler(B) & -cavaler(C).
cavaler(B) -> -cavaler(A) & -cavaler(C).
cavaler(C) -> -cavaler(B) & -cavaler(A).

valet(A) -> -valet(B) & -valet(C).
valet(B) -> -valet(A) & -valet(C).
valet(C) -> -valet(B) & -valet(A).

spion(A) -> -spion(B) & -spion(C).
spion(B) -> -spion(A) & -spion(C).
spion(C) -> -spion(B) & -spion(A).

%Sunt cavaler.
cavaler(A) -> cavaler(A).
valet(A) -> -cavaler(A).
spion(A) -> cavaler(A) | -cavaler(A).

cavaler(B) -> cavaler(A).
valet(B) -> -cavaler(A).
spion(B) -> cavaler(A) | -cavaler(A).

cavaler(C) -> spion(C).
valet(C) -> -spion(C).
spion(C) -> spion(C) | -spion(C).

end_of_list.

```

Folosind Prover9 pentru a indeplini cerinta, programul a obtinut o singura solutie:

```
% Proof 1 at 0.00 (+ 0.01) seconds.
% Length of proof is 52.
% Level of proof is 17.
% Maximum clause weight is 8.
% Given clauses 38.

4 -cavaler(x) & -valet(x) -> spion(x) # label(non_clause).  [assumption
   ].
5 cavaler(A) -> -cavaler(B) & -cavaler(C) # label(non_clause).  [
   assumption].
6 cavaler(B) -> -cavaler(A) & -cavaler(C) # label(non_clause).  [
   assumption].
8 valet(A) -> -valet(B) & -valet(C) # label(non_clause).  [assumption].
9 valet(B) -> -valet(A) & -valet(C) # label(non_clause).  [assumption].
11 spion(A) -> -spion(B) & -spion(C) # label(non_clause).  [assumption].
12 spion(B) -> -spion(A) & -spion(C) # label(non_clause).  [assumption].
17 cavaler(B) -> cavaler(A) # label(non_clause).  [assumption].
18 valet(B) -> -cavaler(A) # label(non_clause).  [assumption].
20 cavaler(C) -> spion(C) # label(non_clause).  [assumption].
23 cavaler(A) & spion(B) & valet(C) # label(non_clause) # label(goal).
   [goal].
27 cavaler(x) | valet(x) | spion(x).  [clausify(4)].
28 -cavaler(A) | -cavaler(B).  [clausify(5)].
29 -cavaler(A) | -cavaler(C).  [clausify(5)].
30 -cavaler(B) | -cavaler(C).  [clausify(6)].
```

```
31 -valet(A) | -valet(B).  [clausify(8)].
32 -valet(A) | -valet(C).  [clausify(8)].
33 -valet(B) | -valet(C).  [clausify(9)].
34 -spion(A) | -spion(B).  [clausify(11)].
35 -spion(A) | -spion(C).  [clausify(11)].
36 -spion(B) | -spion(C).  [clausify(12)].
37 -cavaler(B) | cavaler(A).  [clausify(17)].
38 -valet(B) | -cavaler(A).  [clausify(18)].
39 -cavaler(C) | spion(C).  [clausify(20)].
40 -cavaler(A) | -spion(B) | -valet(C).  [deny(23)].
41 -valet(A) | cavaler(B) | spion(B).  [resolve(31,b,27,b)].
42 -valet(C) | cavaler(A) | spion(A).  [resolve(32,a,27,b)].
43 -valet(C) | cavaler(B) | spion(B).  [resolve(33,a,27,b)].
44 -cavaler(A) | cavaler(B) | spion(B).  [resolve(38,a,27,b)].
45 -cavaler(A) | -spion(B) | cavaler(C) | spion(C).  [resolve(40,c,27,b)
   ].
46 cavaler(B) | spion(B) | cavaler(A) | spion(A).  [resolve(41,a,27,b)].
47 cavaler(A) | spion(A) | cavaler(C) | spion(C).  [resolve(42,a,27,b)].
48 cavaler(B) | spion(B) | cavaler(C) | spion(C).  [resolve(43,a,27,b)].
49 cavaler(B) | cavaler(A) | spion(A) | -spion(C).  [resolve(46,b,36,a)
   ].
50 cavaler(B) | cavaler(C) | spion(C) | -cavaler(A).  [resolve(48,b,45,b
   ),merge(e),merge(f)].
51 cavaler(B) | cavaler(C) | spion(C) | -spion(A).  [resolve(48,b,34,b)
   ].
52 cavaler(B) | cavaler(C) | spion(C) | cavaler(A).  [resolve(51,d,47,b)
   ,merge(e),merge(f)].
53 cavaler(B) | cavaler(C) | cavaler(A) | spion(A).  [resolve(52,c,49,d)
   ,merge(d),merge(e)].
54 cavaler(B) | cavaler(C) | cavaler(A) | -spion(C).  [resolve(53,d,35,a
   )].
55 cavaler(B) | cavaler(C) | cavaler(A).  [resolve(54,d,52,c),merge(d),
   merge(e),merge(f)].
56 cavaler(C) | cavaler(A).  [resolve(55,a,37,a),merge(c)].
57 cavaler(C) | cavaler(B) | spion(C).  [resolve(56,b,50,d),merge(c)].
58 cavaler(C) | cavaler(B) | spion(B).  [resolve(56,b,44,a)].
59 cavaler(C) | cavaler(B) | -spion(C).  [resolve(58,c,36,a)].
61 cavaler(C) | cavaler(B).  [resolve(59,c,57,c),merge(c),merge(d)].
62 cavaler(C) | -cavaler(A).  [resolve(61,b,28,b)].
63 cavaler(C).  [resolve(62,b,56,b),merge(b)].
64 spion(C).  [back_unit_del(39),unit_del(a,63)].
65 -cavaler(B).  [back_unit_del(30),unit_del(b,63)].
66 -cavaler(A).  [back_unit_del(29),unit_del(b,63)].
67 spion(A).  [back_unit_del(49),unit_del(a,65),unit_del(b,66),unit_del(
   d,64)].
69 $F.  [back_unit_del(35),unit_del(a,67),unit_del(b,64)].
```

Concluzii: Presupunem ca omul imbracat in albastru este valet, de aici reiese ca omul in rosu este spion si verde este cavaler, dar propozitia 3 contrazice acest lucru.

Presupunem ca omul imbracat in albastru este cavaler, de aici reiese ca rosu este spion si verde este valet.

Nu presupunem niciodata ca cineva este spion.

2. Departament Store

O persoana care intra intr-un magazin poate avea una din urmatoarele atributii: client, casier, contabil, om de serviciu si manager. Aceste meserii sunt respectate cu strictete de Ana, Bianca, Conroy, David si Evans. Avem urmatoarele conditii:

(a) Casierul si managerul au fost colegi de camera in facultate.

(b) Clientul este burlac.

(c) Evans si Ana au doar intalniri bazate pe afaceri.

(d) Conroy a fost dezamagit cand sotia i-a spus ca managerul nu ii mareste salariul.

(e) David va fi singura tinta dupa ce contabilul si casierul se casatoresc.

Ce pozitie ocupa fiecare persoana in acest magazin?

Codul pentru informatiile care reies din textul dat:

```
1
2  formulas(assumptions).
3
4  hasJob(x, client) | hasJob(x, casier) | hasJob(x,contabil) | hasJob(x,
       omdeserviciu) | hasJob(x,manager).
5  hasJob(Ana, x) | hasJob(Bianca, x) | hasJob(Conroy, x) | hasJob(David, x
       ) | hasJob(Evans, x).
6
7  diffPeople(x,y)->diffPeople(y,x).
8  -diffPeople(x,x).
9
10 diffJobs(x,y)->diffJobs(y,x).
11 -diffJobs(x,x).
12
13 hasJob(x,y) & hasJob(z,y) -> -diffPeople(x,z).
14 hasJob(x,y) & hasJob(x,z) -> -diffJobs(y,z).
```

```
15
16 (male(x) & -female(x)) | (female(x) & -male(x)).
17
18 couple(x,y)->(male(x) & female(y)) | (male(y) & female(x)).
19 couple(x,y)->couple(y,x).
20 couple(x,y)-> diffPeople(x,y).
21
22 burlac(x) -> -couple(x,y) & male(x).
23 onlyBusinessContact(x,y)->onlyBusinessContact(y,x).
24 onlyBusinessContact(x,y) -> -couple(x,y).
25 couple(x,y) -> -onlyBusinessContact(x,y).
26
27 diffJobs(client,casier).
28 diffJobs(client,contabil).
29 diffJobs(client,omdeserviciu).
30 diffJobs(client,manager).
31 diffJobs(casier,contabil).
32 diffJobs(casier,omdeserviciu).
33 diffJobs(casier,manager).
34 diffJobs(contabil,omdeserviciu).
35 diffJobs(contabil,manager).
36 diffJobs(omdeserviciu,manager).
37
38 diffPeople(Ana,Bianca).
39 diffPeople(Ana,Conroy).
40 diffPeople(Ana,David).
41 diffPeople(Ana,Evans).
42 diffPeople(Bianca,Conroy).
43 diffPeople(Bianca,David).
44 diffPeople(Bianca,Evans).
45 diffPeople(Conroy,David).
46 diffPeople(Conroy,Evans).
47 diffPeople(David,Evans).
48
49 female(Ana) & female(Bianca) & male(Conroy) & male(David) & male(Evans).
50 hasJob(x, casier) & hasJob(y, manager) -> -onlyBusinessContact(x,y).
51 hasJob(x, client) -> burlac(x).
52 onlyBusinessContact(Evans,Ana).
53 onlyBusinessContact(Ana,Evans).
54 -burlac(Conroy).
55
56 -hasJob(Conroy, manager).
57 -hasJob(Conroy, client).
58 -hasJob(Conroy,contabil).
59 -hasJob(Conroy,casier).
60
61 hasJob(x,contabil) & hasJob(y,casier) -> couple(x,y).
62 -hasJob(David, contabil).
63 -hasJob(David, casier).
64 hasJob(x,contabil) & hasJob(y,casier)-> -onlyBusinessContact(x,David) |
      -onlyBusinessContact(y,David).
65
66 end_of_list.
67
68 formulas(goals).
69 hasJob(Ana,manager) & hasJob(Bianca,casier) & hasJob(Conroy,omdeserviciu
      ) & hasJob(David,client) & hasJob(Evans,contabil).
70 end_of_list.
71
```

Folosind Prover9 pentru a indeplini cerinta, programul a obtinut urmatoarea solutie:

```
% Proof 1 at 0.01 (+ 0.01) seconds.
% Length of proof is 83.
% Level of proof is 16.
% Maximum clause weight is 18.
% Given clauses 189.

1 diffPeople(x,y) -> diffPeople(y,x) # label(non_clause).  [assumption].
3 hasJob(x,y) & hasJob(z,y) -> -diffPeople(x,z) # label(non_clause).  [
    assumption].
4 hasJob(x,y) & hasJob(x,z) -> -diffJobs(y,z) # label(non_clause).  [
    assumption].
5 male(x) & -female(x) | female(x) & -male(x) # label(non_clause).  [
    assumption].
6 couple(x,y) -> male(x) & female(y) | male(y) & female(x) # label(
    non_clause).  [assumption].
9 burlac(x) -> -couple(x,y) & male(x) # label(non_clause).  [assumption
    ].
11 onlyBusinessContact(x,y) -> -couple(x,y) # label(non_clause).  [
    assumption].
13 female(Ana) & female(Bianca) & male(Conroy) & male(David) & male(
    Evans) # label(non_clause).  [assumption].
14 hasJob(x,casier) & hasJob(y,manager) -> -onlyBusinessContact(x,y) #
    label(non_clause).  [assumption].
15 hasJob(x,client) -> burlac(x) # label(non_clause).  [assumption].
16 hasJob(x,contabil) & hasJob(y,casier) -> couple(x,y) # label(
    non_clause).  [assumption].
18 hasJob(Ana,manager) & hasJob(Bianca,casier) & hasJob(Conroy,
    omdeserviciu) & hasJob(David,client) & hasJob(Evans,contabil) #
    label(non_clause) # label(goal).  [goal].
19 -hasJob(x,client) | burlac(x).  [clausify(15)].
21 -burlac(x) | male(x).  [clausify(9)].
24 hasJob(x,client) | hasJob(x,casier) | hasJob(x,contabil) | hasJob(x,
    omdeserviciu) | hasJob(x,manager).  [assumption].
25 hasJob(Ana,x) | hasJob(Bianca,x) | hasJob(Conroy,x) | hasJob(David,x)
     | hasJob(Evans,x).  [assumption].
26 -diffPeople(x,y) | diffPeople(y,x).  [clausify(1)].
30 -hasJob(x,y) | -hasJob(z,y) | -diffPeople(x,z).  [clausify(3)].
31 -hasJob(x,y) | -hasJob(x,z) | -diffJobs(y,z).  [clausify(4)].
33 -female(x) | -male(x).  [clausify(5)].
34 -couple(x,y) | male(x) | male(y).  [clausify(6)].
39 -onlyBusinessContact(x,y) | -couple(x,y).  [clausify(11)].
44 diffJobs(casier,contabil).  [assumption].
51 diffPeople(Ana,Conroy).  [assumption].
52 diffPeople(Ana,David).  [assumption].
54 diffPeople(Bianca,Conroy).  [assumption].
57 diffPeople(Conroy,David).  [assumption].
58 diffPeople(Conroy,Evans).  [assumption].
60 female(Ana).  [clausify(13)].
61 female(Bianca).  [clausify(13)].
65 -hasJob(x,casier) | -hasJob(y,manager) | -onlyBusinessContact(x,y).
    [clausify(14)].
66 onlyBusinessContact(Evans,Ana).  [assumption].
67 onlyBusinessContact(Ana,Evans).  [assumption].
68 -hasJob(Conroy,manager).  [assumption].
69 -hasJob(Conroy,client).  [assumption].
70 -hasJob(Conroy,contabil).  [assumption].
71 -hasJob(Conroy,casier).  [assumption].
72 -hasJob(x,contabil) | -hasJob(y,casier) | couple(x,y).  [clausify(16)
    ].
73 -hasJob(David,contabil).  [assumption].
```

```
74 -hasJob(David,casier).  [assumption].
76 -hasJob(Ana,manager) | -hasJob(Bianca,casier) | -hasJob(Conroy,
   omdeserviciu) | -hasJob(David,client) | -hasJob(Evans,contabil).  [
   deny(18)].
78 -hasJob(x,client) | male(x).  [resolve(19,b,21,a)].
83 -hasJob(x,omdeserviciu) | -diffPeople(y,x) | hasJob(y,client) |
   hasJob(y,casier) | hasJob(y,contabil) | hasJob(y,manager).  [resolve
   (30,a,24,d)].
85 -hasJob(x,omdeserviciu) | -diffPeople(x,y) | hasJob(y,client) |
   hasJob(y,casier) | hasJob(y,contabil) | hasJob(y,manager).  [resolve
   (30,b,24,d)].
95 -hasJob(x,casier) | -hasJob(x,contabil).  [resolve(44,a,31,c)].
108 diffPeople(Conroy,Ana).  [resolve(51,a,26,a)].
117 -male(Ana).  [resolve(60,a,33,a)].
118 -male(Bianca).  [resolve(61,a,33,a)].
122 -hasJob(Evans,casier) | -hasJob(Ana,manager).  [resolve(66,a,65,c)].
123 -couple(Evans,Ana).  [resolve(66,a,39,a)].
125 -couple(Ana,Evans).  [resolve(67,a,39,a)].
128 hasJob(Ana,contabil) | hasJob(Bianca,contabil) | hasJob(Evans,
   contabil).  [resolve(73,a,25,d),unit_del(c,70)].
129 hasJob(Ana,casier) | hasJob(Bianca,casier) | hasJob(Evans,casier).
   [resolve(74,a,25,d),unit_del(c,71)].
132 -hasJob(Ana,client).  [ur(78,b,117,a)].
135 -couple(Ana,Bianca).  [ur(34,b,117,a,c,118,a)].
136 -couple(Bianca,Ana).  [ur(34,b,118,a,c,117,a)].
142 -hasJob(Evans,omdeserviciu).  [ur(83,b,58,a,c,69,a,d,71,a,e,70,a,f
   ,68,a)].
143 -hasJob(David,omdeserviciu).  [ur(83,b,57,a,c,69,a,d,71,a,e,70,a,f
   ,68,a)].
150 -hasJob(Ana,omdeserviciu).  [ur(83,b,108,a,c,69,a,d,71,a,e,70,a,f
   ,68,a)].
153 -hasJob(Bianca,omdeserviciu).  [ur(85,b,54,a,c,69,a,d,71,a,e,70,a,f
   ,68,a)].
157 hasJob(Conroy,omdeserviciu).  [resolve(143,a,25,d),unit_del(a,150),
   unit_del(b,153),unit_del(d,142)].
158 hasJob(David,client) | hasJob(David,manager).  [resolve(143,a,24,d),
   unit_del(b,74),unit_del(c,73)].
160 -hasJob(Ana,manager) | -hasJob(Bianca,casier) | -hasJob(David,client
   ) | -hasJob(Evans,contabil).  [back_unit_del(76),unit_del(c,157)].
161 hasJob(Ana,casier) | hasJob(Ana,contabil) | hasJob(Ana,manager).  [
   resolve(150,a,24,d),unit_del(a,132)].
168 hasJob(David,client) | -hasJob(x,manager) | -diffPeople(x,David).  [
   resolve(158,b,30,b)].
170 hasJob(Ana,contabil) | hasJob(Evans,contabil) | -hasJob(x,casier) |
   couple(Bianca,x).  [resolve(128,b,72,a)].
173 hasJob(Ana,casier) | hasJob(Evans,casier) | -hasJob(x,contabil) |
   couple(x,Bianca).  [resolve(129,b,72,b)].
176 -hasJob(Bianca,casier) | hasJob(Ana,contabil) | hasJob(Evans,
   contabil).  [resolve(95,b,128,b)].
182 hasJob(David,client) | hasJob(Ana,casier) | hasJob(Ana,contabil).  [
   resolve(168,b,161,c),unit_del(b,52)].
183 hasJob(Ana,contabil) | hasJob(Evans,contabil) | hasJob(Ana,casier) |
    hasJob(Evans,casier).  [resolve(176,a,129,b)].
192 -hasJob(Ana,manager) | -hasJob(Bianca,casier) | -hasJob(Evans,
   contabil) | hasJob(Ana,casier) | hasJob(Ana,contabil).  [resolve
   (160,c,182,a)].
217 -hasJob(Ana,manager) | -hasJob(Evans,contabil) | hasJob(Ana,casier)
   | hasJob(Ana,contabil) | hasJob(Evans,casier).  [resolve(192,b,129,b
   ),merge(e)].
231 -hasJob(Ana,manager) | hasJob(Ana,casier) | hasJob(Ana,contabil) |
```

```
   hasJob(Evans,casier).  [resolve(217,b,183,b),merge(e),merge(f),merge
   (g)].
80 232 hasJob(Ana,casier) | hasJob(Ana,contabil) | hasJob(Evans,casier).  [
   resolve(231,a,161,c),merge(d),merge(e)].
81 235 hasJob(Ana,casier) | hasJob(Ana,contabil) | -hasJob(Ana,manager).  [
   resolve(232,c,122,a)].
82 239 hasJob(Ana,casier) | hasJob(Ana,contabil).  [resolve(235,c,161,c),
   merge(c),merge(d)].
83 240 hasJob(Ana,casier) | hasJob(Evans,casier).  [resolve(239,b,173,c),
   merge(b),unit_del(c,135)].
84 241 hasJob(Ana,casier) | -hasJob(x,casier) | couple(Ana,x).  [resolve
   (239,b,72,a)].
85 248 hasJob(Ana,casier).  [resolve(241,b,240,b),merge(c),unit_del(b,125)
   ].
86 261 hasJob(Ana,contabil) | hasJob(Evans,contabil).  [resolve(248,a,170,c
   ),unit_del(c,136)].
87 266 -hasJob(Ana,contabil).  [ur(95,a,248,a)].
88 268 -hasJob(Evans,contabil).  [ur(72,b,248,a,c,123,a)].
89 278 $F.  [back_unit_del(261),unit_del(a,266),unit_del(b,268)].
90
```

Concluzii:

(a) Conroy este casatorit si poate sa fie doar om de serviciu.

(b) David este burlac doarece restul sunt casatoriti sau urmeaza, deci David este clientul.

(c) Evans se casatoreste cu Bianca deoarece cu Ana are doar intalniri de afaceri, reiese faptul ca Evans este contabil

(d) Contabilul se casatoreste cu casierul, deci Bianca este casier.

(e) Bianca a fost colega de camera cu managerul, deci Ana este manager.

3. Secret Santa

Cinci angajati ai unei companii stau unul langa celalalt la petrecerea de Secret Santa. Afla detalii despre fiecare (nume, ce porta, departamentul unde lucreaza, cadoul primit, varsta si ce bautura consuma) stiind urmatoarele:

(a) Cody este cel mai tanar angajat.
(b) Persoana care a primit cadou o carte sta exact in stanga persoanei care lucreaza la departamentul HR.
(c) Pe pozitia 5 este persoana care bea suc.
(d) Riley este langa angajatul de 41 de ani.
(e) Angajatul de 35 de ani este undeva la final sau la inceput.
(f) Persoana care poarta rosu este undeva intre peroana care a primit un Mug si persoana care bea soft drink, in aceasta ordine.
(g) Angajatul care bea cafea este exact in stanga angajatului care a primit un Notepad cadou.
(h) Persoana care bea ceai este exact in dreapta persoanei care poarta albastru.
(i) Persoana carea poarta verde este langa persoana de 28 de ani.
(j) Steven este exact in dreapta lui Cody.
(k) Pe pozitia 2 este persoana care bea apa.
(l) Angajatul de la departamentul RD este pe pozitia 3.
(m) Cadoul lui Tyler a fost un Mug.

(n) Cel mai in varsta angajat este pe pozitia 5.

(o) Persoana care bea soft drink este pe pozitia 3.

(p) Riley este langa persoana care a primit cadou o cravata.

(q) Cel mai tanar angajat este undeva intre persoana care bea apa si cea mai in varsta persoana, in aceasta ordine.

(r) Jason este exact in dreapta persoanei care poarta negru.

(s) Cody este langa persoana carea bea soft drink.

(t) Persoana care poarta albastru este undeva in stanga persoanei care lucreaza in departamentul sales.

(u) Angajatul departamentului de IT a primit cadou un Notepad.

(v) Pe pozitia 4 este persoana care bea ceai.

Culorile hainelor sunt: alb, verde, albastru, negru, rosu.

Numele angajatilor este: Steven, Riley, Cody, Tyler, Jason.

Cadourile primite sunt: notepad, mug, ciocolata, cravata, carte.

Departamentele: IT, sales, HR, RD, marketing.

Varsta angalatilor: 23, 41, 35, 50, 28.

Bauturile: cafea, apa, ceai, soft drink, suc.

Codul pentru informatiile care reies din textul dat:

```
formulas(assumptions).

differentFrom(a,b).
 differentFrom(a,c).
 differentFrom(a,d).
 differentFrom(a,e).
 differentFrom(b,c).
 differentFrom(b,d).
 differentFrom(b,e).
 differentFrom(c,d).
 differentFrom(c,e).
 differentFrom(d,e).
 differentFrom(x,y) ->  differentFrom(y,x).

 rightneighbor(a,b).
 rightneighbor(b,c).
 rightneighbor(c,d).
 rightneighbor(d,e).
 -rightneighbor(a,a).
 -rightneighbor(a,c).
 -rightneighbor(a,d).
 -rightneighbor(a,e).
 -rightneighbor(b,a).
 -rightneighbor(b,b).
 -rightneighbor(b,d).
 -rightneighbor(b,e).
 -rightneighbor(c,a).
 -rightneighbor(c,b).
 -rightneighbor(c,c).
 -rightneighbor(c,e).
 -rightneighbor(d,a).
 -rightneighbor(d,b).
 -rightneighbor(d,c).
 -rightneighbor(d,d).
```

```
-rightneighbor(e,a).
-rightneighbor(e,b).
-rightneighbor(e,c).
-rightneighbor(e,d).
-rightneighbor(e,e).
rightneighbor(x,y) | rightneighbor(y,x) <-> neighbor(x,y).

somewhereRight(a,b).
somewhereRight(a,c).
somewhereRight(a,d).
somewhereRight(a,e).
somewhereRight(b,c).
somewhereRight(b,d).
somewhereRight(b,e).
somewhereRight(c,d).
somewhereRight(c,e).
somewhereRight(d,e).
-somewhereRight(a,a).
-somewhereRight(b,a).
-somewhereRight(b,b).
-somewhereRight(c,a).
-somewhereRight(c,b).
-somewhereRight(c,c).
-somewhereRight(d,a).
-somewhereRight(d,b).
-somewhereRight(d,c).
-somewhereRight(d,d).
-somewhereRight(e,a).
-somewhereRight(e,b).
-somewhereRight(e,c).
-somewhereRight(e,d).
-somewhereRight(e,e).

between(b,a,c).
between(b,a,d).
between(b,a,e).
between(c,a,d).
between(c,a,e).
between(c,b,d).
between(c,b,e).
between(d,a,e).
between(d,b,e).
between(d,c,e).
-between(a,a,a).
-between(a,a,b).
-between(a,a,c).
-between(a,a,d).
-between(a,a,e).
-between(a,b,a).
-between(a,b,b).
-between(a,b,c).
-between(a,b,d).
-between(a,b,e).
-between(a,c,a).
-between(a,c,b).
-between(a,c,c).
-between(a,c,d).
-between(a,c,e).
-between(a,d,a).
-between(a,d,b).
```

```
-between(a,d,c).
-between(a,d,d).
-between(a,d,e).
-between(a,e,a).
-between(a,e,b).
-between(a,e,c).
-between(a,e,d).
-between(a,e,e).
-between(b,a,a).
-between(b,a,b).
-between(b,b,a).
-between(b,b,b).
-between(b,b,c).
-between(b,b,d).
-between(b,b,e).
-between(b,c,a).
-between(b,c,b).
-between(b,c,c).
-between(b,c,d).
-between(b,c,e).
-between(b,d,a).
-between(b,d,b).
-between(b,d,c).
-between(b,d,d).
-between(b,d,e).
-between(b,e,a).
-between(b,e,b).
-between(b,e,c).
-between(b,e,d).
-between(b,e,e).
-between(c,a,a).
-between(c,a,b).
-between(c,a,c).
-between(c,b,a).
-between(c,b,b).
-between(c,b,c).
-between(c,c,a).
-between(c,c,b).
-between(c,c,c).
-between(c,c,d).
-between(c,c,e).
-between(c,d,a).
-between(c,d,b).
-between(c,d,c).
-between(c,d,d).
-between(c,d,e).
-between(c,e,a).
-between(c,e,b).
-between(c,e,c).
-between(c,e,d).
-between(c,e,e).
-between(d,a,a).
-between(d,a,b).
-between(d,a,c).
-between(d,a,d).
-between(d,b,a).
-between(d,b,b).
-between(d,b,c).
-between(d,b,d).
-between(d,c,a).
```

```
-between(d,c,b).
-between(d,c,c).
-between(d,c,d).
-between(d,d,a).
-between(d,d,b).
-between(d,d,c).
-between(d,d,d).
-between(d,d,e).
-between(d,e,a).
-between(d,e,b).
-between(d,e,c).
-between(d,e,d).
-between(d,e,e).
-between(e,a,a).
-between(e,a,b).
-between(e,a,c).
-between(e,a,d).
-between(e,a,e).
-between(e,b,a).
-between(e,b,b).
-between(e,b,c).
-between(e,b,d).
-between(e,b,e).
-between(e,c,a).
-between(e,c,b).
-between(e,c,c).
-between(e,c,d).
-between(e,c,e).
-between(e,d,a).
-between(e,d,b).
-between(e,d,c).
-between(e,d,d).
-between(e,d,e).
-between(e,e,a).
-between(e,e,b).
-between(e,e,c).
-between(e,e,d).
-between(e,e,e).

black(x) | blue(x) | green(x) | white(x) | red(x).
cody(x) | steven(x) | jason(x) | tyler(x) | riley(x).
mug(x) | notepad(x) | book(x) | tie(x) | chocolate(x).
IT(x) | RD(x) | HR(x) | marketing(x) | sales(x).
water(x) | tea(x) | juice(x) | coffee(x) | softdrink(x).
age50(x) | age23(x) | age35(x) | age41(x) | age28(x).

black(x) & black(y) -> -differentFrom(x,y).
blue(x) & blue(y) -> -differentFrom(x,y).
green(x) & green(y) -> -differentFrom(x,y).
white(x) & white(y) -> -differentFrom(x,y).
red(x) & red(y) -> -differentFrom(x,y).

cody(x) & cody(y) -> -differentFrom(x,y).
steven(x) & steven(y) -> -differentFrom(x,y).
jason(x) & jason(y) -> -differentFrom(x,y).
tyler(x) & tyler(y) -> -differentFrom(x,y).
riley(x) & riley(y) -> -differentFrom(x,y).

mug(x) & mug(y) -> -differentFrom(x,y).
notepad(x) & notepad(y) -> -differentFrom(x,y).
```

```
book(x) & book(y) -> -differentFrom(x,y).
tie(x) & tie(y) -> -differentFrom(x,y).
chocolate(x) & chocolate(y) -> -differentFrom(x,y).

IT(x) & IT(y) -> -differentFrom(x,y).
RD(x) & RD(y) -> -differentFrom(x,y).
HR(x) & HR(y) -> -differentFrom(x,y).
marketing(x) & marketing(y) -> -differentFrom(x,y).
sales(x) & sales(y) -> -differentFrom(x,y).

water(x) & water(y) -> -differentFrom(x,y).
tea(x) & tea(y) -> -differentFrom(x,y).
juice(x) & juice(y) -> -differentFrom(x,y).
coffee(x) & coffee(y) -> -differentFrom(x,y).
softdrink(x) & softdrink(y) -> -differentFrom(x,y).

age50(x) & age50(y) -> -differentFrom(x,y).
age23(x) & age23(y) -> -differentFrom(x,y).
age35(x) & age35(y) -> -differentFrom(x,y).
age41(x) & age41(y) -> -differentFrom(x,y).
age28(x) & age28(y) -> -differentFrom(x,y).

cody(x) <-> age23(x).
book(x) & HR(y) -> rightneighbor(x,y).
juice(e).
riley(x) & age41(y)-> neighbor(x,y).
age35(a) | age35(e).
red(x) & mug(y) & softdrink(z) -> between(x,y,z).
coffee(x) & notepad(y) -> rightneighbor(x,y).
tea(x) & blue(y) -> rightneighbor(y,x).
green(x) & age28(y)-> neighbor(x,y).
steven(x) & cody(y) -> rightneighbor(y,x).
water(b).
RD(c).
tyler(x) <-> mug(x).
age50(e).
softdrink(c).
riley(x) & tie(y)-> neighbor(x,y).
age23(x) & water(y) & age50(z) -> between (x,y,z).
jason(x) & black(y) -> rightneighbor(y,x).
cody(x) & softdrink(y)-> neighbor(x,y).
blue(x) & sales(y) -> somewhereRight(x,y).
IT(x) <-> notepad(x).
tea(d).
end_of_list.


```

Folosind Mace4 am obtinut urmatoarea solutie:

```
interpretation( 5, [number = 1,seconds = 0], [
    function(a, [0]),
    function(b, [1]),
    function(c, [2]),
    function(d, [3]),
    function(e, [4]),
    relation(HR(_), [0,0,0,1,0]),
    relation(IT(_), [0,1,0,0,0]),
    relation(RD(_), [0,0,1,0,0]),
    relation(age23(_), [0,0,0,1,0]),
    relation(age28(_), [0,0,1,0,0]),
```

```
    relation(age35(_), [1,0,0,0,0]),
    relation(age41(_), [0,1,0,0,0]),
    relation(age50(_), [0,0,0,0,1]),
    relation(black(_), [1,0,0,0,0]),
    relation(blue(_), [0,0,1,0,0]),
    relation(book(_), [0,0,1,0,0]),
    relation(chocolate(_), [0,0,0,0,1]),
    relation(cody(_), [0,0,0,1,0]),
    relation(coffee(_), [1,0,0,0,0]),
    relation(green(_), [0,0,0,1,0]),
    relation(jason(_), [0,1,0,0,0]),
    relation(juice(_), [0,0,0,0,1]),
    relation(marketing(_), [1,0,0,0,0]),
    relation(mug(_), [1,0,0,0,0]),
    relation(notepad(_), [0,1,0,0,0]),
    relation(red(_), [0,1,0,0,0]),
    relation(riley(_), [0,0,1,0,0]),
    relation(sales(_), [0,0,0,0,1]),
    relation(softdrink(_), [0,0,1,0,0]),
    relation(steven(_), [0,0,0,0,1]),
    relation(tea(_), [0,0,0,1,0]),
    relation(tie(_), [0,0,0,1,0]),
    relation(tyler(_), [1,0,0,0,0]),
    relation(water(_), [0,1,0,0,0]),
    relation(white(_), [0,0,0,0,1]),
    relation(differentFrom(_,_), [
        0,1,1,1,1,
        1,0,1,1,1,
        1,1,0,1,1,
        1,1,1,0,1,
        1,1,1,1,0]),
    relation(neighbor(_,_), [
        0,1,0,0,0,
        1,0,1,0,0,
        0,1,0,1,0,
        0,0,1,0,1,
        0,0,0,1,0]),
    relation(rightneighbor(_,_), [
        0,1,0,0,0,
        0,0,1,0,0,
        0,0,0,1,0,
        0,0,0,0,1,
        0,0,0,0,0]),
    relation(somewhereRight(_,_), [
        0,1,1,1,1,
        0,0,1,1,1,
        0,0,0,1,1,
        0,0,0,0,1,
        0,0,0,0,0]),
    relation(between(_,_,_),
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0
]).

```

Concluzii: incepem prin a completa in tabel informatiile concrete

(a) Enunturile (c), (k), (l), (n), (o), (v) stim exact ce sa punem in tabel la fiecare pozitie.

(b) Din (h) ne dam seama ca pe pozitia 3 persoana poarta albastru.

(c) Din (s) stim ca putem sa il punem pe Cody pe pozitia 4 si ca este cel mai tanar angajat (a).

(d) Steven este in dreapta lui Cody adica pe pozitia 5 (j).

(e) Angajatul de pe pozitia 1 bea cafea, fiind singura bautura ramasa si stim ca persoana din dreapta lui a primit cadou un Notepad (g).

(f) Angajatul care a primit un Notepad lucreaza in departamentul IT, deci pozitia 2 (u).

(g) Angajatul de 35 de ani este undeva la margine, ultima pozitie fiind luata rezulta ca acesta este primul (e).

(h) Din (f) reiese ca primul angajat a primit cadou un Mug pentru a se pastra ordinea corecta, deci persoana care poarta rosu este pe 2.

(i) Din (m) stim ca persoana care a primit un Mug este Tyler.

(j) Stim din (t) ca in stanga persoanei care poarta albastru este cineva care lucreaza la departamentul sales, deci pozitia 5.

(k) Din (b) deducem ca persoana de pe pozitia 3 este cea care a primit cadou cartea fiind singura optiune valabila si de aici reiese ca persoana de pe pozitia 4 lucreaza in departamentul de HR.

(l) Cody a primit cadou o cravata si stim din (p) ca Riley este langa el, deci pe pozitia 3.

(m) Riley este langa angajatul de 41 de ani, deci aceasta este varsta de pe pozitia 2 (d) si deci Riley are 28 de ani, fiind sigura optiune ramasa.

(n) Angajatul care poarta verde este langa cel de 28 de ani, deci pe pozita 4.

(o) Jason se afla pe pozitia 2, singurul nume disponibil si stim ca in stanga lui persoana poarta negru (r).

(p) Mai raman de completat la Steven optiunile disponibile, acesta poarta alb si a primit cadou ciocolata.

	Employee #1	Employee #2	Employee #3	Employee #4	Employee #5
Shirt	black	red	blue	green	white
Name	Tyler	Jason	Riley	Cody	Steven
Gift	mug	notepad	book	tie	chocolate
Department	marketing	IT	R&D	HR	sales
Age	35 years	41 years	28 years	23 years	50 years
Drink	coffee	water	soft drink	tea	juice

4. Secret Agent

Problema spionilor se refera la o echipa de 5 spioni, fiecare spion avand o cravata de o anumita culoare, un nume, o tara in care urmeaza sa mearga, un accesoriu, o aptitudine si o varsta. Toate aceste atribute sunt unice pentru fiecare spion.Spionii sunt asezati intr-un sir dupa urmatoarele indicii:

(a) Austin e langa spionul cu cravata neagra.
(b) Maestrul in deghizari este exact in dreapta spionului care are umbrela.
(c) Spionul de 35 de ani se duce in Tripoli.
(d) James este cel mai tanar spion.
(e) Spionul care se duce in Australia este langa spionul care are ca aptitudine parkour-ul.
(f) James este exact in dreapta spionului care are ceas.
(g) Spionul care are umbrela este undeva intre spionul de 40 de ani si Austin, in aceasta ordine.

(h) Stan este langa agentul care se duce in Asia.
(i) Sterling este la unul dintre capete.
(j) Barbatul care poarta cravata rosie are 40 de ani.
(k) Spionul care se duce in America de Sud este exact in stanga spionului de 40 de ani.
(l) Jason este exact in stanga lui Austin.
(m) Experul in condus este langa spionul de 30 de ani.
(n) Spionul de 35 de ani este langa spionul care se duce in Sydney.
(o) Spionul specializat in computer hacking este exact in stanga spionului de 35 de ani.
(p) Spionul care poarta cravata mov este langa spionul specializat in computer hack-ing.
(q) Austin are 30 de ani.
(r) Spionul care are telefon este chiar in stanga spionului care se duce in Africa.
(s) Spionul care are inel este undeva la dreapta spionului care poarta cravata mov.
(t) In pozitia 2 este spionul care poarta cravata verde.
(u) Spionul care merge in Australia este exact in dreapta spionului de 30 de ani.

Cravatele sunt de 5 culori: negru, albastru, verde, mov si rosu.

Numele lor sunt: Austin, James, Jason, Stan si Sterling.

Tarile in care vor merge sunt: Australia, Brazilia, Germania, Libia si Rusia.

Accesoriile lor sunt: ceas, stilou, telefon, inel si umbrela.

Aptitudinile lor sunt: computer hacking, deghizare, condus, arte martiale si parkour.

Varstele lor sunt: 25 de ani, 30 de ani, 35 de ani, 40 de ani si 45 de ani.

Codul pentru informatiile care reies din textul dat:

```
formulas(assumptions).

differentFrom(a,b).
 differentFrom(a,c).
 differentFrom(a,d).
 differentFrom(a,e).
 differentFrom(b,c).
 differentFrom(b,d).
 differentFrom(b,e).
 differentFrom(c,d).
 differentFrom(c,e).
 differentFrom(d,e).
 differentFrom(x,y) ->  differentFrom(y,x).

 rightneighbor(a,b).
 rightneighbor(b,c).
 rightneighbor(c,d).
 rightneighbor(d,e).
 -rightneighbor(a,a).
 -rightneighbor(a,c).
 -rightneighbor(a,d).
 -rightneighbor(a,e).
 -rightneighbor(b,a).
 -rightneighbor(b,b).
 -rightneighbor(b,d).
 -rightneighbor(b,e).
 -rightneighbor(c,a).
 -rightneighbor(c,b).
 -rightneighbor(c,c).
```

```
-rightneighbor(c,e).
-rightneighbor(d,a).
-rightneighbor(d,b).
-rightneighbor(d,c).
-rightneighbor(d,d).
-rightneighbor(e,a).
-rightneighbor(e,b).
-rightneighbor(e,c).
-rightneighbor(e,d).
-rightneighbor(e,e).
rightneighbor(x,y) | rightneighbor(y,x) <-> neighbor(x,y).

somewhereRight(a,b).
somewhereRight(a,c).
somewhereRight(a,d).
somewhereRight(a,e).
somewhereRight(b,c).
somewhereRight(b,d).
somewhereRight(b,e).
somewhereRight(c,d).
somewhereRight(c,e).
somewhereRight(d,e).
-somewhereRight(a,a).
-somewhereRight(b,a).
-somewhereRight(b,b).
-somewhereRight(c,a).
-somewhereRight(c,b).
-somewhereRight(c,c).
-somewhereRight(d,a).
-somewhereRight(d,b).
-somewhereRight(d,c).
-somewhereRight(d,d).
-somewhereRight(e,a).
-somewhereRight(e,b).
-somewhereRight(e,c).
-somewhereRight(e,d).
-somewhereRight(e,e).

between(b,a,c).
between(b,a,d).
between(b,a,e).
between(c,a,d).
between(c,a,e).
between(c,b,d).
between(c,b,e).
between(d,a,e).
between(d,b,e).
between(d,c,e).
-between(a,a,a).
-between(a,a,b).
-between(a,a,c).
-between(a,a,d).
-between(a,a,e).
-between(a,b,a).
-between(a,b,b).
-between(a,b,c).
-between(a,b,d).
-between(a,b,e).
-between(a,c,a).
-between(a,c,b).
```

```
-between(a,c,c).
-between(a,c,d).
-between(a,c,e).
-between(a,d,a).
-between(a,d,b).
-between(a,d,c).
-between(a,d,d).
-between(a,d,e).
-between(a,e,a).
-between(a,e,b).
-between(a,e,c).
-between(a,e,d).
-between(a,e,e).
-between(b,a,a).
-between(b,a,b).
-between(b,b,a).
-between(b,b,b).
-between(b,b,c).
-between(b,b,d).
-between(b,b,e).
-between(b,c,a).
-between(b,c,b).
-between(b,c,c).
-between(b,c,d).
-between(b,c,e).
-between(b,d,a).
-between(b,d,b).
-between(b,d,c).
-between(b,d,d).
-between(b,d,e).
-between(b,e,a).
-between(b,e,b).
-between(b,e,c).
-between(b,e,d).
-between(b,e,e).
-between(c,a,a).
-between(c,a,b).
-between(c,a,c).
-between(c,b,a).
-between(c,b,b).
-between(c,b,c).
-between(c,c,a).
-between(c,c,b).
-between(c,c,c).
-between(c,c,d).
-between(c,c,e).
-between(c,d,a).
-between(c,d,b).
-between(c,d,c).
-between(c,d,d).
-between(c,d,e).
-between(c,e,a).
-between(c,e,b).
-between(c,e,c).
-between(c,e,d).
-between(c,e,e).
-between(d,a,a).
-between(d,a,b).
-between(d,a,c).
-between(d,a,d).
```

```
-between(d,b,a).
-between(d,b,b).
-between(d,b,c).
-between(d,b,d).
-between(d,c,a).
-between(d,c,b).
-between(d,c,c).
-between(d,c,d).
-between(d,d,a).
-between(d,d,b).
-between(d,d,c).
-between(d,d,d).
-between(d,d,e).
-between(d,e,a).
-between(d,e,b).
-between(d,e,c).
-between(d,e,d).
-between(d,e,e).
-between(e,a,a).
-between(e,a,b).
-between(e,a,c).
-between(e,a,d).
-between(e,a,e).
-between(e,b,a).
-between(e,b,b).
-between(e,b,c).
-between(e,b,d).
-between(e,b,e).
-between(e,c,a).
-between(e,c,b).
-between(e,c,c).
-between(e,c,d).
-between(e,c,e).
-between(e,d,a).
-between(e,d,b).
-between(e,d,c).
-between(e,d,d).
-between(e,d,e).
-between(e,e,a).
-between(e,e,b).
-between(e,e,c).
-between(e,e,d).
-between(e,e,e).

black(x) | blue(x) | green(x) | purple(x) | red(x).
austin(x) | james(x) | jason(x) | stan(x) | sterling(x).
australia(x) | brazil(x) | germany(x) | libya(x) | russia(x).
clock(x) | pen(x) | phone(x) | ring(x) | umbrella(x).
computerhacking(x) | disguise(x) | driving(x) | martialarts(x) |
  parkour(x).
age25(x) | age30(x) | age35(x) | age40(x) | age45(x).

black(x) & black(y) -> -differentFrom(x,y).
blue(x) & blue(y) -> -differentFrom(x,y).
green(x) & green(y) -> -differentFrom(x,y).
purple(x) & purple(y) -> -differentFrom(x,y).
red(x) & red(y) -> -differentFrom(x,y).

austin(x) & austin(y) -> -differentFrom(x,y).
james(x) & james(y) -> -differentFrom(x,y).
```

```
jason(x) & jason(y) -> -differentFrom(x,y).
stan(x) & stan(y) -> -differentFrom(x,y).
sterling(x) & sterling(y) -> -differentFrom(x,y).

australia(x) & australia(y) -> -differentFrom(x,y).
brazil(x) & brazil(y) -> -differentFrom(x,y).
germany(x) & germany(y) -> -differentFrom(x,y).
libya(x) & libya(y) -> -differentFrom(x,y).
russia(x) & russia(y) -> -differentFrom(x,y).

clock(x) & clock(y) -> -differentFrom(x,y).
pen(x) & pen(y) -> -differentFrom(x,y).
phone(x) & phone(y) -> -differentFrom(x,y).
ring(x) & ring(y) -> -differentFrom(x,y).
umbrella(x) & umbrella(y) -> -differentFrom(x,y).

computerhacking(x) & computerhacking(y) -> -differentFrom(x,y).
disguise(x) & disguise(y) -> -differentFrom(x,y).
driving(x) & driving(y) -> -differentFrom(x,y).
martialarts(x) & martialarts(y) -> -differentFrom(x,y).
parkour(x) & parkour(y) -> -differentFrom(x,y).

age25(x) & age25(y) -> -differentFrom(x,y).
age30(x) & age30(y) -> -differentFrom(x,y).
age35(x) & age35(y) -> -differentFrom(x,y).
age40(x) & age40(y) -> -differentFrom(x,y).
age45(x) & age45(y) -> -differentFrom(x,y).

austin(x) & black(y) -> neighbor(x,y).
disguise(x) & umbrella(y) -> rightneighbor(y,x).
age35(x) <-> libya(x).
james(x) <-> age25(x).
australia(x) & parkour(y) -> neighbor(x,y).
james(x) & clock(y) -> rightneighbor(y,x).
umbrella(x) & age40(y) & austin(z) -> between (x,y,z).
stan(x) & russia(y) -> neighbor(x,y).
sterling(a) | sterling(e).
red(x) <-> age40(x).
brazil(x) & age45(y) -> rightneighbor(x,y).
jason(x) & austin(y) -> rightneighbor(x,y).
driving(x) & age30(y) -> neighbor(x,y).
age35(x) & australia(y) -> neighbor(x,y).
computerhacking(x) & age35(y) -> rightneighbor(x,y).
purple(x) & computerhacking(y) -> neighbor(x,y).
austin(x) <-> age30(x).
phone(x) & libya(y) -> rightneighbor(x,y).
ring(x) & purple(y) -> somewhereRight(y,x).
green(b).
australia(x) & age30(y) -> rightneighbor(y,x).

end_of_list.
```

Folosind Mace4 am obtinut urmatoarea solutie:

```
interpretation( 5, [number = 1,seconds = 0], [
    function(a, [0]),
    function(b, [1]),
    function(c, [2]),
    function(d, [3]),
    function(e, [4]),
```

```
    relation(age25(_), [0,0,0,1,0]),
    relation(age30(_), [0,0,1,0,0]),
    relation(age35(_), [0,0,0,0,1]),
    relation(age40(_), [1,0,0,0,0]),
    relation(age45(_), [0,1,0,0,0]),
    relation(austin(_), [0,0,1,0,0]),
    relation(australia(_), [0,0,0,1,0]),
    relation(black(_), [0,0,0,1,0]),
    relation(blue(_), [0,0,0,0,1]),
    relation(brazil(_), [1,0,0,0,0]),
    relation(clock(_), [0,0,1,0,0]),
    relation(computerhacking(_), [0,0,0,1,0]),
    relation(disguise(_), [0,0,1,0,0]),
    relation(driving(_), [0,1,0,0,0]),
    relation(germany(_), [0,0,1,0,0]),
    relation(green(_), [0,1,0,0,0]),
    relation(james(_), [0,0,0,1,0]),
    relation(jason(_), [0,1,0,0,0]),
    relation(libya(_), [0,0,0,0,1]),
    relation(martialarts(_), [1,0,0,0,0]),
    relation(parkour(_), [0,0,0,0,1]),
    relation(pen(_), [1,0,0,0,0]),
    relation(phone(_), [0,0,0,1,0]),
    relation(purple(_), [0,0,1,0,0]),
    relation(red(_), [1,0,0,0,0]),
    relation(ring(_), [0,0,0,0,1]),
    relation(russia(_), [0,1,0,0,0]),
    relation(stan(_), [1,0,0,0,0]),
    relation(sterling(_), [0,0,0,0,1]),
    relation(umbrella(_), [0,1,0,0,0]),
    relation(differentFrom(_,_), [
        0,1,1,1,1,
        1,0,1,1,1,
        1,1,0,1,1,
        1,1,1,0,1,
        1,1,1,1,0]),
    relation(neighbor(_,_), [
        0,1,0,0,0,
        1,0,1,0,0,
        0,1,0,1,0,
        0,0,1,0,1,
        0,0,0,1,0]),
    relation(rightneighbor(_,_), [
        0,1,0,0,0,
        0,0,1,0,0,
        0,0,0,1,0,
        0,0,0,0,1,
        0,0,0,0,0]),
    relation(somewhereRight(_,_), [
        0,1,1,1,1,
        0,0,1,1,1,
        0,0,0,1,1,
        0,0,0,0,1,
        0,0,0,0,0]),
    relation(between(_,_,_),
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0
]).

```

Concluzii: incepem prin a completa in tabel informatiile concrete:

(a) Din (t) punem verde pe pozitia 2.

(b) Verificam pozitiile lui Austin: stim ca trebuie ca unul dintre spioni sa fie intre spionul de 40 de ani si Austin (g), deci Austin se afla pe pozitiile 3, 4 sau 5. Austin are 30 de ani (q) si stim ca spionul care merge in Australia este in dreapta sa (u), deci Austin se afla pe pozitiile 3 sau 4. Spionul de 35 de ani este langa spionul care merge in Australia (n), ceea ce inseamna ca spionul care merge in Australia este pe pozitia 4, iar Austin este pe pozitia 3. Spionul de 35 de ani este pe pozitia 5 si merge in Libia (c). Jason este in stanga lui Austin, deci pe pozitia 2 (l).

(c) Austin sta langa spionul cu cravata neagra, deci pe pozitia 4 se afla cravata neagra (a).

(d) Spionul care are telefon este in stanga celui care merge in Libia, deci acesta se afla pe pozitia 4 (r).

(e) Spionul care e specializat in computer hacking este in stanga celui de 35 de ani, deci acesta se afla tot pe pozitia 4 (o).

(f) Austin se afla langa spionul specializat in condus, deci cel din urma este pe pozitia 2 (m).

(g) Spionul care are umbrela este intre cel de 40 de ani si Austin, deci cel cu umbrela este pe pozitia 2 si cel de 40 de ani este pe pozitia 1 (g).

(h) Cel de 40 de ani are cravata rosie, deci cravata rosie este pe pozitia 1 (j).

(i) Spionul cu cravata mov este langa cel specializat in computer hacking, deci pe pozitiile 3 sau 5 (p). Spionul care are inel este undeva in dreapta celui cu cravata mov (s), deci cel cu cravata mov trebuie sa fie pe pozitia a 3-a, iar cel cu inel pe pozitia 5.

(j) Prin eliminare, spionul cu cravata albastra este pe pozitia a 5-a.

(k) James este in dreapta spionului care are ceas (f), deci James este pe pozitia 4, iar cel care are ceas este pe pozitia 3.

(l) James are 25 de ani(d). Prin eliminare, cel care are 45 de ani este pe pozitia a 2-a. Prin eliminare, spionul care are stilou este pe pozitia 1.

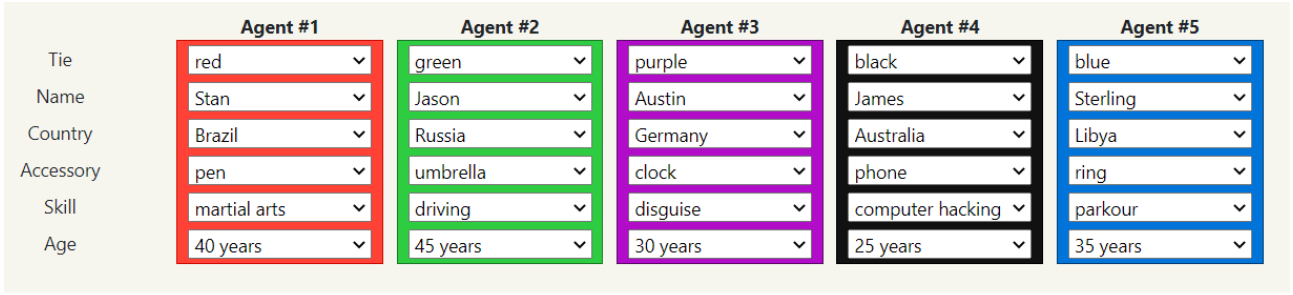
(m) Cel care merge in Brazilia este in stanga celui de 45 de ani (k), deci pe pozitia 1. Spionul specializat in deghizare este in dreapta celui cu umbrela, deci pe pozitia 3.

(n) Spionul care se duce in Australia este langa cel specializat in parkour (e), deci cel din urma este pe pozitia 5.

(o) Stan este langa spionul care merge in Rusia (h), deci Stan este pe pozitia 1, iar cel care merge in Rusia este pe pozitia 2.

(p) Sterling este la unul dintre capete, deci pe pozitia 5 (i).

(q) Prin eliminare, cel care merge in Germania este pe pozitia 3, iar cel care e specializat in arte martiale este pe pozitia 1.



	Agent #1	Agent #2	Agent #3	Agent #4	Agent #5
Tie	red	green	purple	black	blue
Name	Stan	Jason	Austin	James	Sterling
Country	Brazil	Russia	Germany	Australia	Libya
Accessory	pen	umbrella	clock	phone	ring
Skill	martial arts	driving	disguise	computer hacking	parkour
Age	40 years	45 years	30 years	25 years	35 years

Chapter 3

A3: Planning

Pentru aceasta parte am ales sa implementez un CoffeeShop.

In aceata cafenea un client care vine trebuie sa isi aleaga din sortimentul propus bautura pe care si-o doreste (Ristretto, Espresso, Long Black, Cappuccino, Latte, Flat White, Caramel Macchiato, Frappe) si modul in care doreste sa realizeze plata (cash sau card).

Definirea domeniului cuprinde toate actiunile posibile in cafenea:

1. (openCoffeShop) - deschiderea cafenelei;
2. (Client ?x) - exista un client;
3. (Full) - casierul este ocupat;
4. (Comanda) - comanda s-a realizat;
5. (Plata) - plata s-a realizat;
6. (testR) - clientul vrea cafea Ristretto - acelasi lucru si pentru restul predicatelor de acest tip;
7. (modulCash) - clientul alege plata cash - la fel si pentru card;
8. (ALegeCafea ?x) - tipurile de cafea pe care un client le poate alege;

Urmatoarea actiune defineste deschiderea cafenelei:

```
   (:action openCoffeeShop
    :parameters ()
    :precondition (not (openCoffeeShop))
    :effect (openCoffeShop)
)

```

Pentru un client nou:

```
   (:action newClient
    :parameters (?x)
    :precondition (and (openCoffeeShop) (not(Client ?x))     (not(Full)))
    :effect (and (Client ?x) (Full))
)

```

Pentru a alege modalitatea de plata (in cazul platii cu cardul se schimba doar variabila de test):

```
   (:action plataCash
    :parameters (?x)
    :precondition (and (openCoffeeShop) (testCash ?x) (Client ?x))
    :effect (and (modulCash ?x) (increase (total-cost) 1))
)

```

Pentru alegerea cafelei dorite (la fiecare sortiment exista o variabila de testare diferita):

```
     (:action chooseLatte
    :parameters (?x)
    :precondition (and (openCoffee) (testL ?x) (Client ?x))
    :effect (Latte ?x)
)

```

Problemele pe care le-am propus pentru acest domeniu sunt urmatoarele:

1. In cafenea exista un client care isi comanda un Latte si doreste sa plateasca cu cardul.

```
    (define (problem coffee)
  (:domain CoffeeShop)
  (:objects c1)
  (:init (testL c1) (testCard c1) (=(total-cost)0))
  (:goal (AlegeCafea c1))
  (:metric minimize(total-cost))
)

```

2. Pentru urmatoarea problema in cafenea exista 2 clienti. Primul vrea un Latte si plateste cu cardul, iar al doilea client vrea un Flat White si plata sa fie cash.

```
    (define (problem coffee)
  (:domain CoffeeShop)
  (:objects c11 c12)
  (:init (testL c11) (testCard c11) (testFW c12) (testCash c12) (=(
  total-cost)0))
  (:goal (and (AlegeCafea c11) (AlegeCafea c12)))
  (:metric minimize(total-cost))
)

```

3. In acest caz exista 3 clienti. Primul vrea un Caramel Macchiato si sa plateasca cu cardul. Al doilea vrea un Frappe si plata tot cu cardul, iar clientul 3 vrea un Espresso si sa plateasca cash.

```
    (define (problem coffee)
  (:domain CoffeeShop)
  (:objects c11 c12 c13)
  (:init (testCM c11) (testCard c11) (testF c12) (testCard c12)    (
  testE c13) (testCash c13) (=(total-cost)0))
  (:goal (and (AlegeCafea c11) (AlegeCafea c12) (AlegeCafea c13)))
  (:metric minimize(total-cost))
)

```

4. Ultima problema propune ca un client sa comande si un Latte, dar si un Cappuccino platind intreaga comanda cu cardul.

```
    (define (problem coffee)
  (:domain CoffeeShop)
  (:objects c1)
  (:init (testL c1) (testC c1) (testCard c1) (=(total-cost)0))
  (:goal (AlegeCafea c1))
  (:metric minimize(total-cost))
)

```