Artificial intelligence - Project 2
- Logica propozitionala si a predicatelor-

Galis Laurentiu

8/12/2020

1 Introducere

Pentru realizarea acestui proiect au fost nevoie de cunostinte legate de satisfacerea conditiilor si logica propozitionala a predicatelor.

Logica propozitionala este cea mai simpla logica

- Simbolurile True si False - sunt propozitii atomice
- Simbolurile P1, P2 etc sunt propozitii atomice
- Daca S este o propozitie, ¬S este o propozitie (negation)
- Daca S1 si S2 sunt propozitii, S1 & S2 este o propozitie (conjunction)
- Daca S1 si S2 sunt propozitii, S1 | S2 este o propozitie (disjunction)
- Daca S1 si S2 sunt propozitii, S1 -> S2 este o propozitie (implication)
- Daca S1 si S2 sunt propozitii, S1 <-> S2 este o propozitie (biconditional)

Proiectul este format din doua probleme de tipul **zebra puzzle** in care trebuie sa ajungem la o solutie unica pe baza unor statement-uri. Problemele pe care le-am ales se numesc **Einstein's riddle, Dogs for Adoption si The Fourth Deduction**.

1.1 Cerintele problemelor

1.1.1 Einstein's riddle

In aceasta problema este vorba despre 5 case (a,b,c,d,e) puse una langa alta fiecare avand cate o culoare diferita. In fiecare casa traieste o persoana cu o nationalitate diferita si fiecare persoana are o bautura, brand de tigara favorita, si un animal diferit. Fiecare persoana are cate un animal diferit, bea o bautura diferita si are un brand de tigari favorit diferit fata de ceilalti. Intrebarea problemei este cine detine pestele?(animalele folosite sunt enumerate in statement-urile problemei pe care le voi descrie in capitolul urmator).

1.1.2 Dogs for Adoption

In aceasta problema este vorba despre 5 (a,b,c,d,e) persoane care sunt la un pet shop si stau la coada unul langa altul pentru a adopta cate un caine. Fiecare persoana are o imbracaminte cu o culoare diferita, un nume, profesie si hobby diferit. Acestea adopta cate un caine cu un nume si rasa diferita.

1.1.3 The Fourth Deduction

I usually keep my notes in case files that live in the drawers of a now rather full filing cabinet, but recently I had cause to keep a couple of them out on my desk in anticipation of writing one of my well-received reports. Unfortunately, Holmes had left the window open and then gone out, and so when I returned home I found the various pieces of paper in a state of disarray, strewn across the floor. This was most frustrating, as the cases had taken place long enough ago that I was unable to entirely remember the details of each one. What I was, however, able to gather from the loose pieces of paper was as follows:

- The three cases were to be entitled The Adventure of the Broken Table, The Adventure of the Frozen Lake, and The Adventure of the Moving Statue.
- There were three victims, called John Bell, Sarah Doyle and Mark Robinson
- There were three perpetrators, called Juliet Lane, Charlotte Green and Peter Watkins.
- The crimes committed were robbery, fraud and murder.

Moreover, between the two of us, Holmes and I were able to remember a few key facts about the cases:

- Juliet Lane defrauded her uncle.

- Mark Robinson's body was found under a lake that had frozen over.
- Peter Watkins was caught because of a splinter in his hand which he'd obtained from the jagged edge of a dining room table at the crime scene.

Combining these facts together, we were able to deduce the victim, perpetrator and crime committed in each of the three cases. Can you do the same?

2 Solutie conceptuala

Pentru rezolvarea acestor probleme fara ajutorul unor tool-uri splecifice cum ar fi Prover9 sau Mace4 este nevoie de construirea unui tabel si trecerea tuturor informatiilor in el in functie de constrangerile enumerate in enuntul problemei. Formularea acestor enunturi pot duce la generarea mai multor modele sau la un singur model. Problemele alese de mine genereaza un singur model acest lucru crescand complexitatea problemei deoarece nu fiecare enunt duce la rezolvarea altui enunt.

2.1 Constrangerile problemelor

Enunturile care duc la generarea unui singur model la fiecare problema sunt:

2.1.1 Einstein's riddle

- britanicul locuieste in casa rosie
- suedezul are un caine
- danezul bea ceai
- casa verde este in stanga casei albe
- proprietarul casei verde bea cafea
- persoana care fumeaza Pall Mall are pasari
- proprietarul carei galbene fumeaza Dunhill
- proprietarul casei din mijloc bea lapte
- norvegianul traieste in prima casa
- proprietarul care fumeaza blends traieste langa proprietarul care are pisici
- proprietarul care are un cal traieste langa proprietarul care fumeaza Dunhill
- proprietarul care fumeaza blueMaster bea bere
- germanul fumeaza Prince
- norvegianul locuieste labga casa albastra
- proprietarul care fumeaza blends are un vecin care bea apa

2.1.2 Dogs for Adoption

- persoana care adopta Rottweiler e pe prima pozitie
- asistenta e langa persoana careia ii place Hiking
- persoana careia ii place Jogging-ul e la unul din capete
- persoana care adopta Dachshund e in dreapta persoanei care poarta galben
- persoana care in adopta pe Rex e pe a 4-a pozitie
- chelnerul e intre omul care poarta negru si pictor
- Rex e adoptat de Kevin
- persoanei careia ii place cititul e intre persoana careia ii place gatitul si careia ii plae cititul
- Ryan adopta Poodle
- persoana careia ii place desenatul i intre persoan carei ii place gatitul si cea careia ii place cititul
- persoana care adopta Beagle-ul e la unul dintre capete
- pe a 4-a pozitie se pe pozitia a se afla persoana careia ii place Hiking-ul
- Bob e actor

- persoana care adopta Apollo e pe a 5-a pozitie
- persoana care adopta Chihuaua e langa persoana care adopta Dachshund
- persoana care poarta albastru e pe a 4-a pozitie
- Hank il adopta pe Spyke
- persoana care poarta rosu e langa persoan care adopta Dachshund
- persoana care poarta alb e intre persoana care adopta pe Luke si cel careia ii place Jogging-ul
- Designer-ul poarta alb

2.2 Prove9 si Mace4

Prove9 este un tool folosit pentru a demonstra teoreme bazate pe logica propozitionala si FOL(First Oeder Logic). Mace4 este folosit pentru a genera modele pe baza logicii propozitionale sau FOL.
Aceste tool-uri sunt create pentru a ilustra concepte cum ar fi:

- rezolutie
- validitate
- satisfiabilitate
- modele
- Conjunctive Normal Form(CNF)

Prover9 cauta probe, iar Mace4 cauta contraexemple. Propozitiile cu care ele opereaza pot fi scrise in logica propozitionala, FOL sau ecuatii logice. .

3 Implementare

Ambele probleme au aproape aceeasi idee de implementare diferenta dintre ele fiind constrangerile si numarul acestora. Cu cat numarul constrangerilor este mai mare sau complexitatea acestora este mai mare, cu atat si problema va fi mult mai complexa.

3.1 Reprezentare in FOL

In primul rand pentru a reprezenta in FOL o problema bazata pe logica propozitionala trebuie luata in considerare dimensiunea domeniului. Abmele probleme au aceeasi dimensiune (domain size:5) si aceeasi denumire (a,b,c,d,e).
Constantele comune ambelor probleme sunt: a,b,c,d,e.
Predicatele comune ambelor probleme sunt:

- differentFrom(x,y): x e diferit fata de y
- rightneighbor(x,y): y e vecinul drept lui x
- leftneighbor(x,y): y e vecinul stang lui x
- neighbor(x,y): x si y sunt vecini

3.1.1 Einstein's riddle

Predicatele folosite la aceasta problema, pe langa cele de sus sunt:

- brit(x) | swede(x) | dane(x) | norwegian(x) | german(x): nationalitatea proprietarilor
- red(x) | white(x) | yellow(x) | blue(x) | green(x): culoarea caselor
- dog(x) | bird(x) | cat(x) | horse(x) | fish(x): animalele proprietarilor
- tea(x) | coffe(x) | milk(x) | beer(x) | water(x): bauturile preferate
- pallMall(x) | dunhill(x) | blueMaster(x) | prince(x) | blend(x): tigarile folosite

Code:

```
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).

%relatie de simetrie
 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).
 rightneighbor(y,x) -> leftneighbor(x,y).

% predicatele fiecarei case

% nationalitatea
 brit(x) | swede(x) | dane(x) | norwegian(x) | german(x).
% culoarea
 red(x) | white(x) | yellow(x) | blue(x) | green(x).
% animale
 dog(x) | bird(x) | cat(x) | horse(x) | fish(x).
% bauturi
 tea(x) | coffe(x) | milk(x) | beer(x) | water(x).
% tigari
 pall_mall(x) | dunhill(x) | blueMaster(x) | prince(x) | blend(x).

% fiecare predicat se aplica unei singure case

 brit(x) & brit(y) -> -differentFrom(x,y).
 swede(x) & swede(y) -> -differentFrom(x,y).
 dane(x) & dane(y) -> -differentFrom(x,y).
 norwegian(x) & norwegian(y) -> -differentFrom(x,y).
 german(x) & german(y) -> -differentFrom(x,y).

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

 dog(x) & dog(y) -> -differentFrom(x,y).
 bird(x) & bird(y) -> -differentFrom(x,y).
 cat(x) & cat(y) -> -differentFrom(x,y).
 horse(x) & horse(y) -> -differentFrom(x,y).
 fish(x) & fish(y) -> -differentFrom(x,y).

 tea(x) & tea(y) -> -differentFrom(x,y).
 coffe(x) & coffe(y) -> -differentFrom(x,y).
 milk(x) & milk(y) -> -differentFrom(x,y).
 beer(x) & beer(y) -> -differentFrom(x,y).
 water(x) & water(y) -> -differentFrom(x,y).

 pall_mall(x) & pall_mall(y) -> -differentFrom(x,y).
 dunhill(x) & dunhill(y) -> -differentFrom(x,y).
 blueMaster(x) & blueMaster(y) -> -differentFrom(x,y).
 prince(x) & prince(y) -> -differentFrom(x,y).
 blend(x) & blend(y) -> -differentFrom(x,y).

% Hint-urile
 brit(x) <-> red(x).
 swede(x) <-> dog(x).
 dane(x) <-> tea(x).
 white(x) & green(y) -> leftneighbor(x,y).
 green(x) <-> coffe(x).
 pall_mall(x) <-> bird(x).
 yellow(x) <-> dunhill(x).
 milk(c).
 norwegian(a).
 blend(x) & cat(y) -> neighbor(x,y).
 horse(x) & dunhill(y) -> neighbor(x,y).
 blueMaster(x) <-> beer(x).
 german(x) <-> prince(x).
 norwegian(x) & blue(y) -> neighbor(x,y).
 blend(x) & water(y) -> neighbor(x,y).
```

Explicarea scrierii statement-urilor:

- Pentru a arata ca o constanta locuieste la o anumita casa, am specificat predicatul urmat de constanta care se afla la o anumita poitie(De ex: norvegianul traieste in prima casa -> norwegian(a))
- Pentru a arata ca doua predicate apartin aceluiasi domeniu am specificat predicatele urmat de aceeasi constanta(De ex: suedezul are un caine -> swede(x) <-> dog(x))
- Pentru a arata ca doua constante sunt vecini am specificat predicatele acestora si am folosti predicatul definit mai sus (neighbor)(De ex: norvegianul locuieste langa casa albastra -> norwegian(x) & blue(y) -> neighbor(x,y))
- Pentru a arata ca o constanta se afla la stanga sau la dreapta altei constante am folosit predicatese specifice constantelor plus constanta definita mai sus(leftneghbor sau rightneighbor)(De ex: casa verde este in stanga casei albe -> white(x) & green(y) -> leftneighbor(x,y))

3.1.2 Dogs for Adoption

Predicatele folosite la aceasta problema, pe langa cele de sus sunt:

- black(x) | white(x) | yellow(x) | blue(x) | red(x): culoarea hainelor
- bob(x) | dylan(x) | hank(x) | kevin(x) | ryan(x): numele persoanelor
- apollo(x) | jake(x) | luke(x) | rex(x) | spyke(x): numele cainilor
- beagle(x) | chihuaua(x) | dachshund(x) | poodle(x) | rottweiler(x): rasele cainilor
- actor(x) | designer(x) | nurse(x) | painter(x) | waiter(x): job-urile persoanelor
- cooking(x) | drawing(x) | hiking(x) | jogging(x) | reading(x): hobby-urile persoanelor

Code:

```
 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).

%relatie de simetrie
 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).
 rightneighbor(y,x) -> leftneighbor(x,y).

% predicatele fiecarei case

% nationalitatea
 brit(x) | swede(x) | dane(x) | norwegian(x) | german(x).
% culoarea
 red(x) | white(x) | yellow(x) | blue(x) | green(x).
% animale
 dog(x) | bird(x) | cat(x) | horse(x) | fish(x).
% bauturi
 tea(x) | coffe(x) | milk(x) | beer(x) | water(x).
% tigari
```

```
 pall_mall(x) | dunhill(x) | blueMaster(x) | prince(x) | blend(x).

% fiecare predicat se aplica unei singure case

 brit(x) & brit(y) -> -differentFrom(x,y).
 swede(x) & swede(y) -> -differentFrom(x,y).
 dane(x) & dane(y) -> -differentFrom(x,y).
 norwegian(x) & norwegian(y) -> -differentFrom(x,y).
 german(x) & german(y) -> -differentFrom(x,y).

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

 dog(x) & dog(y) -> -differentFrom(x,y).
 bird(x) & bird(y) -> -differentFrom(x,y).
 cat(x) & cat(y) -> -differentFrom(x,y).
 horse(x) & horse(y) -> -differentFrom(x,y).
 fish(x) & fish(y) -> -differentFrom(x,y).

 tea(x) & tea(y) -> -differentFrom(x,y).
 coffe(x) & coffe(y) -> -differentFrom(x,y).
 milk(x) & milk(y) -> -differentFrom(x,y).
 beer(x) & beer(y) -> -differentFrom(x,y).
 water(x) & water(y) -> -differentFrom(x,y).

 pall_mall(x) & pall_mall(y) -> -differentFrom(x,y).
 dunhill(x) & dunhill(y) -> -differentFrom(x,y).
 blueMaster(x) & blueMaster(y) -> -differentFrom(x,y).
 prince(x) & prince(y) -> -differentFrom(x,y).
 blend(x) & blend(y) -> -differentFrom(x,y).

% Hint-urile
 brit(x) <-> red(x).
 swede(x) <-> dog(x).
 dane(x) <-> tea(x).
 white(x) & green(y) -> rightneighbor(y,x).
 green(x) <-> coffe(x).
 pall_mall(x) <-> bird(x).
 yellow(x) <-> dunhill(x).
 milk(c).
 norwegian(a).
 blend(x) & cat(y) -> neighbor(x,y).
 horse(x) & dunhill(y) -> neighbor(x,y).
 blueMaster(x) <-> beer(x).
 german(x) <-> prince(x).
 norwegian(x) & blue(y) -> neighbor(x,y).
 blend(x) & water(y) -> neighbor(x,y).
```

Explicarea scrierii statement-urilor:

- Pentru a arata ca o constanta sta la o anumita pozitie, am specificat predicatul urmat de constanta care se afla la o anumita poitie(De ex: persoana care adopta Rottweiler e pe prima pozitie -> rottweiler(a))

- Pentru a arata ca doua predicate apartin celeiasi constante am specificat ca doua constante nu sunt diferite una fata de cealalta(-differentFrom(x,y))(De ex: Ryan adopta Poodle -> ryan(x) & poodle(y) -> -differentFrom(x,y))
- Pentru a arata ca doua constante sunt vecini am specificat predicatele acestora si am folosti predicatul definit mai sus (neighbor)(De ex: asistenta e langa persoana careia ii place Hiking -> nurse(x) & hiking(y) -> neighbor(x,y))
- Pentru a arata ca o constanta se afla la stanga sau la dreapta altei constante am folosit predicatese specifice constantelor plus constanta definita mai sus(leftneghbor sau rightneighbor)(De ex: persoana care adopta Dachshund e in dreapta persoanei care poarta galben -> dachshund(x) & yellow(y) -> rightneighbor(y,x))
- Pentru a specifica ca o constanta se afla intre alte doua constante am specificate predicatele respective si am folosit operatorul SAU pentru a verifica daca acea constanta este in stanga sau dreapta uneia dintre celelalte constante(De ex: chelnerul e intre omul care poarta negru si pictor -> (black(x) & waiter(y) -> rightneighbor(x,y)) | (waiter(y) & painter(z) -> rightneighbor(y,z)))
- Pentru a specifica daca o constanta se afla la una din capetele domeniului am folosit operatorul SAU(De ex: persoana careia ii place Jogging-ul e la unul din capete -> jogging(a) | jogging(e))

3.1.3 The Third Deduction

Predicatele folosite la aceasta problema, pe langa cele de sus sunt:

- TheAdventureoftheBrokenTable(x) | TheAdeventureoftheFrozenLake(x) | TheAdventureoftheMovingStatue(x): cazurile
- JohnBell(x) | SarahDoyle(x) | MarkRobinson(x): victimele
- JulietLane(x) | CharlotteGreen(x) | PeterWatkins(x): faptasii
- robbery(x) | fraud(x) | murder(x): crimele comise

Code:

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

TheAdventureoftheBrokenTable(x) | TheAdeventureoftheFrozenLake(x) | TheAdventureoftheMovingStatue(x).
%victime
JohnBell(x) | SarahDoyle(x) | MarkRobinson(x).
%faptasi
JulietLane(x) | CharlotteGreen(x) | PeterWatkins(x).
%crime comise
robbery(x) | fraud(x) | murder(x).

TheAdventureoftheBrokenTable(x)& TheAdventureoftheBrokenTable(y) -> -differentFrom(x,y).
TheAdeventureoftheFrozenLake(x) & TheAdeventureoftheFrozenLake(y) -> -differentFrom(x,y).
TheAdventureoftheMovingStatue(x) & TheAdventureoftheMovingStatue(y) -> -differentFrom(x,y).

JohnBell(x) & JohnBell(y) -> -differentFrom(x,y).
SarahDoyle(x) & SarahDoyle(y) -> -differentFrom(x,y).
MarkRobinson(x) & MarkRobinson(y) -> -differentFrom(x,y).

JulietLane(x) & JulietLane(y) -> -differentFrom(x,y).
CharlotteGreen(x) & CharlotteGreen(y) -> -differentFrom(x,y).
PeterWatkins(x) & PeterWatkins(y) -> -differentFrom(x,y).
```

```

robbery(x) & robbery(y) -> -differentFrom(x,y).
fraud(x) & fraud(y) -> -differentFrom(x,y).
murder(x) & murder(y) -> -differentFrom(x,y).

JulietLane(x) <-> fraud(x).

MarkRobinson(x) <-> TheAdeventureoftheFrozenLake(x).
MarkRobinson(x) <-> murder(x).

PeterWatkins(x) <-> TheAdventureoftheBrokenTable(x).

fraud(x) <-> TheAdventureoftheMovingStatue(x).
```

4 Rezultate

Pentru rezolvarea abmebor probleme am folosit tool-ul Mace4.
Pentru a putea intelege ce afiseaza Mace4 trebuie stiu in primul rand ca valoarea ’0’ inseamna False si ’1’ inseamna True. Mace4 genereaza modele, asadar acest tool afiseaza dimensiunea domeniului (5), numarul de modele generate si secundele(number = 1,seconds = 0) si o lista in care se afla functii care reproduc rezultatul final. Toate acestea sunt argumentele functiei interpretation(interpretation(5, [number = 1,seconds = 0], [...]). Intelegerea functiilor:

- function(): e formata dintr-o constantela si pozitia lui din domeniu
- relation(): e formata dintr-un predicat si valoarea de adevar a fiecarei constante din acesta(De ex: relation(beer(_), [0,0,0,1,0] -> predicatul beer are constanta d adevarata). Asadar, aceasta functie afiseaza modelul final generand valoarea de adevar a constantelor fiecarui predicat.

Output-urile fiecarei probleme sunt descrise mai jos.

4.1 Einstein’s riddle

In final, Mace4 afiseaza raspunsul cerintei: "Cine detine pestele?" se poate vedea foarte usor in output-ul generat.(Pestele e detinut de proprietarul care se afla in casa d(verde), mai exact de german).

Output:

```
interpretation( 5, [number = 1,seconds = 0], [
    function(a, [0]),
    function(b, [1]),
    function(c, [2]),
    function(d, [3]),
    function(e, [4]),
    relation(beer(_), [0,0,0,0,1]),
    relation(bird(_), [0,0,1,0,0]),
    relation(blend(_), [0,1,0,0,0]),
    relation(blue(_), [0,1,0,0,0]),
    relation(blueMaster(_), [0,0,0,0,1]),
    relation(brit(_), [0,0,1,0,0]),
    relation(cat(_), [1,0,0,0,0]),
    relation(coffe(_), [0,0,0,1,0]),
    relation(dane(_), [0,1,0,0,0]),
    relation(dog(_), [0,0,0,0,1]),
    relation(dunhill(_), [1,0,0,0,0]),
    relation(fish(_), [0,0,0,1,0]),
    relation(german(_), [0,0,0,1,0]),
    relation(green(_), [0,0,0,1,0]),
    relation(horse(_), [0,1,0,0,0]),
    relation(milk(_), [0,0,1,0,0]),
    relation(norwegian(_), [1,0,0,0,0]),
    relation(pall_mall(_), [0,0,1,0,0]),
    relation(prince(_), [0,0,0,1,0]),
    relation(red(_), [0,0,1,0,0]),
    relation(swede(_), [0,0,0,0,1]),
    relation(tea(_), [0,1,0,0,0]),
    relation(water(_), [1,0,0,0,0]),
    relation(white(_), [0,0,0,0,1]),
    relation(yellow(_), [1,0,0,0,0]),
    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(leftneighbor(_,_), [
        0,0,0,0,0,
        1,0,0,0,0,
        0,1,0,0,0,
        0,0,1,0,0,
        0,0,0,1,0]),
    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])]).
```

4.2 Dogs for Adoption

Output:

```
interpretation( 5, [number = 1,seconds = 0], [
    function(a, [0]),
    function(b, [1]),
    function(c, [2]),
    function(d, [3]),
    function(e, [4]),
    relation(actor(_), [1,0,0,0,0]),
    relation(apollo(_), [0,0,0,0,1]),
    relation(beagle(_), [0,0,0,0,1]),
    relation(black(_), [1,0,0,0,0]),
    relation(blue(_), [0,0,0,1,0]),
    relation(bob(_), [1,0,0,0,0]),
    relation(chihuaua(_), [0,0,1,0,0]),
    relation(cooking(_), [0,0,0,0,1]),
    relation(dachshund(_), [0,0,0,1,0]),
    relation(designer(_), [0,1,0,0,0]),
    relation(drawing(_), [0,1,0,0,0]),
    relation(dylan(_), [0,0,0,0,1]),
    relation(hank(_), [0,0,1,0,0]),
    relation(hiking(_), [0,0,0,1,0]),
    relation(jake(_), [0,1,0,0,0]),
    relation(jogging(_), [1,0,0,0,0]),
    relation(kevin(_), [0,0,0,1,0]),
    relation(luke(_), [1,0,0,0,0]),
    relation(nurse(_), [0,0,0,0,1]),
    relation(painter(_), [0,0,0,1,0]),
    relation(poodle(_), [0,1,0,0,0]),
    relation(reading(_), [0,0,1,0,0]),
    relation(red(_), [0,0,0,0,1]),
    relation(rex(_), [0,0,0,1,0]),
    relation(rottweiler(_), [1,0,0,0,0]),
    relation(ryan(_), [0,1,0,0,0]),
    relation(spyke(_), [0,0,1,0,0]),
    relation(waiter(_), [0,0,1,0,0]),
    relation(white(_), [0,1,0,0,0]),
    relation(yellow(_), [0,0,1,0,0]),
    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(between(_,_,_), [0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1
```

4.3 The Third Deductionn

Output:

```
interpretation( 3, [number = 1,seconds = 0], [
    function(a, [0]),
    function(b, [1]),
    function(c, [2]),
    relation(CharlotteGreen(_), [0,0,1]),
    relation(JohnBell(_), [0,1,0]),
    relation(JulietLane(_), [0,1,0]),
    relation(MarkRobinson(_), [0,0,1]),
    relation(PeterWatkins(_), [1,0,0]),
    relation(SarahDoyle(_), [1,0,0]),
    relation(TheAdeventureoftheFrozenLake(_), [0,0,1]),
    relation(TheAdventureoftheBrokenTable(_), [1,0,0]),
    relation(TheAdventureoftheMovingStatue(_), [0,1,0]),
    relation(fraud(_), [0,1,0]),
    relation(murder(_), [0,0,1]),
    relation(robbery(_), [1,0,0]),
    relation(differentFrom(_,_), [
        0,1,1,
        1,0,1,
        1,1,0])]).
```

5 Concluzie

In concluzie, acest proect a a avut scopul de a aprofunda ecuatiile logice cu operatorii specifici ,logica prpozitionala, invatarea si folosirea metodei FOL, rezolvarea mai usoara a problemelor logice si folosirea tool-urilor specifice Prover9 si Mace4.