Department of Computer Science
Technical University of Cluj-Napoca

Artificial Intelligence

Laboratory activity

Name: Hulea Andrei-Florin
Group: 30235
Email: andrei.hulea1@gmail.com

Search Assignment Partner
Name: Giurgiu Dorian-Mihai
Group: 30234
Email: mihaigiurgiu37@gmail.com

Logics Assignment Partner
Name: Borca Adrian-Lucian
Group: 30235
Email: adrianborca@gmail.com

Contents

Chapter 1

A1: Search

1.1 Introduction

Obiectivul acestui proiect este de a intelege cum functioneaza algoritmii de cautare, cat si implementarea, testarea si compararea acestora in jocul Pacman. Scopul acestor algoritmi este de a gasi cel mai scurt drum intre doua puncte, intr-un mod cat mai eficient si rapid, mai exact in cazul acestui proiect, de a-l ajuta pe Pacman sa parcurga un drum sau sa manance toate boabele cat mai repede.Acestia sunt evaluati prin complexitatea lor in timp si spatiu, dar si prin optimalitate si completitudine.

In acest proiect am implementat trei noi algoritmi de cautare pe langa cei studiati la laborator, cat si euristici pentru eficientizarea unora dintre acestia. De asemenea, am extins functionalitatile jocului prin adaugarea portalelor. Atunci cand Pacman intra intr-un portal, el este teleportat la locatia altui portal, de unde el isi poate continua drumul.

1.2 Algoritmi de cautare

1.2.1 Weighted A*

Weighted A* este foarte asemanator cu A*, diferenta dintre acestea fiind calculul costului euristicii. Acesta inmulteste valoarea costului euristicii cu epsilon, scopul fiind gasirea celei mai scurte cai cand valoarea euristica este mica. Avand in vedere aceste lucruri, Weighted A* sacrifica optimalitatea pentru viteza, fiind cu mult mai eficient decat A* in cazurile cand nodul final se afla pe un nivel superficial. Acesta este folosit de obicei impreuna cu euristici ce au fost create special pentru gasirea distantei minime de pe nivelele superficiale. Deci, se poate deduce ca Weighted A* este A* ce foloseste euristici consistente pentru epsilon si se poate observa ca daca epsilon are o valoare mare algoritmul se comporta ca un algoritm greedy, daca e 0 ca ucs si daca e 1 ca si A*.

```
def weightedAStarSearch(problem, heuristic=nullHeuristic):
    "*** YOUR CODE HERE ***"
    epsilon = 1.2
    #epsilon=1 => a*
    #epsilon=0 => ucs
    #epsilon=infinity => greedy
    startingNode = problem.getStartState()
    if problem.isGoalState(startingNode):
        return []

    visitedNodes = []
```

```
    pQueue = util.PriorityQueue()
    pQueue.push((startingNode, [], 0), 0)

    while not pQueue.isEmpty():

        currentNode, actions, prevCost = pQueue.pop()

        if currentNode not in visitedNodes:
            visitedNodes.append(currentNode)

            if problem.isGoalState(currentNode):
                return actions

            for nextNode, action, cost in problem.getSuccessors(currentNode):
                newAction = actions + [action]
                newCostToNode = prevCost + cost
                heuristicCost = newCostToNode + epsilon * heuristic(nextNode, problem)
                pQueue.push((nextNode, newAction, newCostToNode), heuristicCost)
    util.raiseNotDefined()
```

1.2.2 Iterative Deepening Depth-First Search

Daca am avea o problema de complexitate in spatiu infinita, algoritmii DFS si BFS nu ar fi suficient de eficienti. In cazul DFS-ului, complexitatea acestuia in spatiu este O(adancimea cautarii), deci daca avem un graf foarte adanc acesta este foarte probabil ca acesta sa nu gaseasca cea mai scurta cale sau o solutie optimala. In cazul BFS-ului, pentru o problema unde graful este foarte adanc acesta are nevoie de prea multa memorie deoarece cauta prima oara toate nodurile vecine nodului curent. Iterative Deepening Depth First Search este un hibrid al celor doi. Functioneaza ca un DFS pana la o limita de adancime pe care o crestem la fiecare iteratie pana gasim solutia. De asemenea, daca limita de adancime este o valoare mare, algoritmul se va comporta foarte asemanator cu un DFS. Pe cazul problemei noastre de gasire a drumului, o valoare mica a limitei de adancime se dovedeste a fi foarte ineficienta deoarece se expandeaza foarte multe noduri.

```
def iterativeDeepeningDFS(problem):
    from searchAgents import mazeDistance
    currentNode = problem.getStartState()
    if problem.isGoalState(currentNode):
        return []

    s = util.Stack()
    depth = 9999

    while True:

        visitedNodes = []
        s.push((problem.getStartState(), [], 0))
        currentNode, actions, prevCost = s.pop()
        visitedNodes.append(currentNode)

        while not problem.isGoalState(currentNode):
```

```
            for newNode, action, cost in problem.getSuccessors(currentNode):
                if newNode not in visitedNodes:
                    if prevCost + cost <= depth:
                        newAction = actions + [action]
                        newCost = prevCost + cost
                        s.push((newNode, newAction, newCost))
                        visitedNodes.append(newNode)
            if s.isEmpty():
                break
            else:
                currentNode, actions, prevCost = s.pop()

        if problem.isGoalState(currentNode):
            return actions
        else:
            depth = depth + 1
```

1.2.3 Iterative Deepening A*

Acest algoritm este o varianta a IDDFS-ului care imprumuta ideea de a folosi o euristica ca sa evalueze costul ramas pentru a ajunge la tinta. Deoarece este un DFS, memoria utilizata de acesta este mai mica decat la A*, dar comparativ cu alte cautari iterative de adancire, acesta se concentreaza pe explorarea nodurilor care par cele mai promitatorare si nu cauta la aceeasi adancime peste tot in arborele de cautare. Tosusi, asemenea IDDFS-ului, IDA* va explora uneori aceleasi noduri de mai multe ori

Asadar, la fiecare iteratie se va efectua o cautare DFS, eliminand ramura cand costul sau total este mai mare decat limita. Limita creste la fiecare iteratie a algoritmului.

```
def iterativeDeepeningAStar(problem):
    from searchAgents import mazeDistance

    currentNode = problem.getStartState()
    if problem.isGoalState(currentNode):
        return []

    s = util.Stack()
    depth = 0

    while True:

        visitedNodes = []

        s.push((problem.getStartState(), [], 0))
        currentNode, actions, prevCost = s.pop()
        visitedNodes.append(currentNode)

        while not problem.isGoalState(currentNode):
            for newNode, action, cost in problem.getSuccessors(currentNode):
                if newNode not in visitedNodes:
                    if prevCost + cost <= depth:
                        newAction = actions + [action]
```

```
                newCost = prevCost + cost
                s.push((newNode, newAction, newCost))
                visitedNodes.append(newNode)
        if s.isEmpty():
            break
        else:
            currentNode, actions, prevCost = s.pop()

    if problem.isGoalState(currentNode):
        return actions
    else:
        depth = depth + mazeDistance(currentNode, problem.goal, problem.state)
```

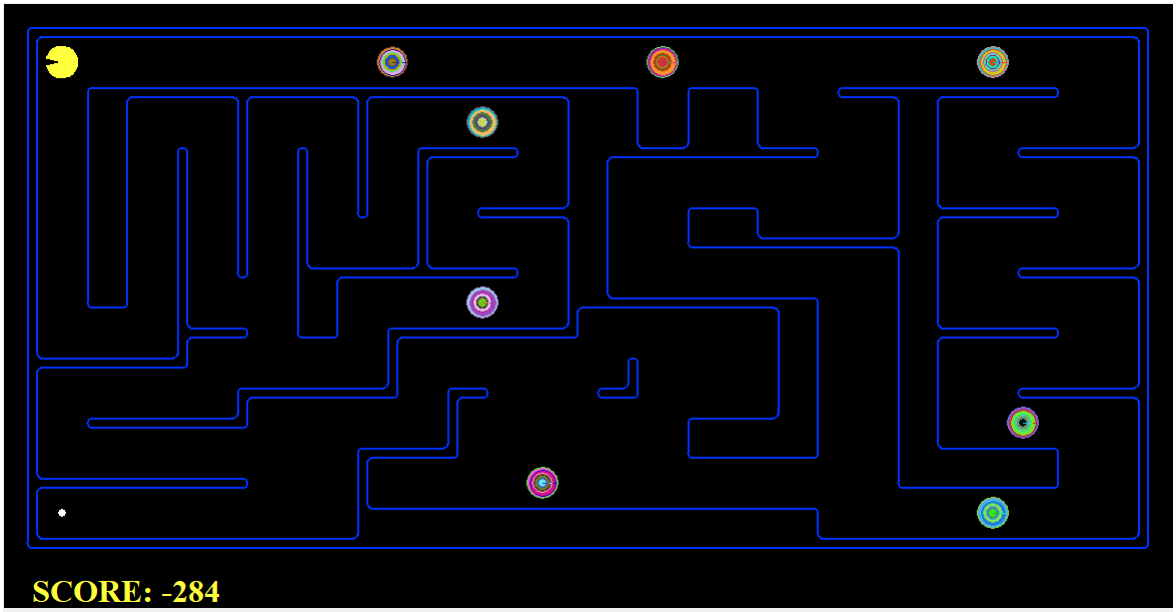
1.2.4 Comparatie Algoritmi

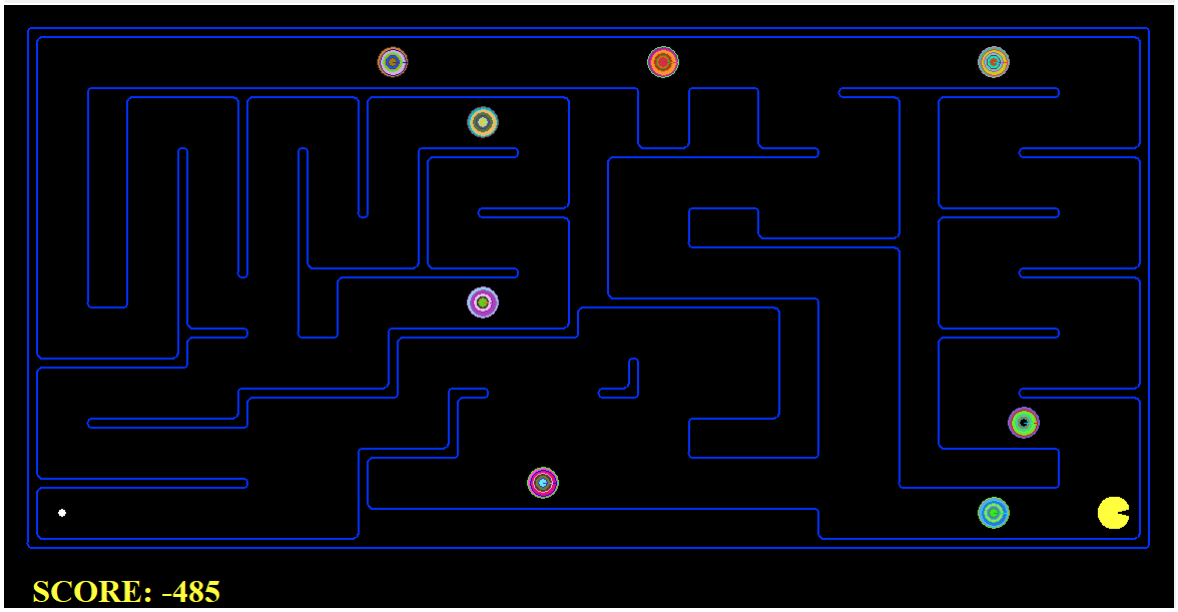
Algoritm	Layout	Expanded Nodes	Cost	Score	Time
DFS	tinyMaze	15	10	500	0.0
DFS	mediumMaze	146	130	380	0.0
DFS	bigMaze	390	210	300	0.0
BFS	tinyMaze	15	8	502	0.0
BFS	mediumMaze	269	68	442	0.0
BFS	bigMaze	620	210	300	0.0
UCS	tinyMaze	15	8	502	0.0
UCS	mediumMaze	269	68	442	0.0
UCS	bigMaze	620	210	300	0.0
A*	tinyMaze	15	8	502	0.0
A*+manhattan	tinyMaze	14	8	502	0.0
A*	mediumMaze	269	68	442	0.0
A*+manhattan	mediumMaze	221	68	442	0.0
A*	bigMaze	620	210	300	0.0
A*+manhattan	bigMaze	549	210	300	0.0
IDDFS	tinyMaze	14	10	500	0.0
IDDFS	mediumMaze	144	130	380	0.1
IDDFS	bigMaze	390	210	300	0.3
IDA*	tinyMaze	28	10	500	0.0
IDA*	mediumMaze	193	68	380	0.1
IDA*	bigMaze	367	210	300	0.3

1.3 Warp tunnels

Am modificat framework-ul Pacman astfel incat sa putem folosi teleportarea prin intermediul portalelor. Pentru adaugarea unui portal trebuie modificat manual harta de layout si adaugat caracterul 'w' la locatia dorita. Astfel, cand Pacman ajunge la locatia unui portal, acesta va fi teleportat la locatia altui portal. Problemele de cautare functioneaza cand avem 2 portale dar pot fi adaugate oricate portale deoarece cand avem mai mult de 2, odata ce Pacman intra intr-un portal el va fi teleportat la alt portal ales random. In interfata jocului portalele sunt reprezentate prin mai multe cercuri suprapuse cu culori random, care isi schimba constant culorile, incercand sa simuleze un efect de 'swirl'.

In primul rand am modificat layout.py astfel incat sa recunoasca caracterul 'w' pe harta ca fiind portal, dupa care am modificat fisierul graphicsDisplay.py pentru ca acestea sa fie afisate pe harta. Dupa acestea am modificat game.py pentru a putea considera portalele ca obiecte pe harta precum mancarea si fantomele. Ultimele modificari au fost facute in pacman.py, unde am adaugat conditie pentru intrarea in portal, si searchAgents.py, unde am adaugat conditie pentru sincronizarea iesirii din portal.





1.4 Concluzii

In urma acestui proiect am reimplementat clasicii algoritmi BFS si DFS, observand cum se descurca comparativ cu alti algoritmi mai complecsi precum UCS si variatiile A*-urlui. Am resusit sa observam in mod practic diferenta dintre cautarea neinformata si cea informata cu aplicarea euristicilor. De asemenea am vazut cum functioneaza framework-ul jocului Pacman si cum trebuie modificat pentru a realiza modificarile propuse.

Incercari nereusite: Ms Pacman cu Pacman junior , algoritmul de cautare D* Lite

Chapter 2

A2: Logics

2.1 Introduction

Obiectivul acestui proiect este de a intelege si a implementa diferite probleme de logica cu ajutorul programelor de demonstrat teoreme Prover9 si Mace4. Pe parcursul proiectului am implementat diferite probleme de logica cu scopul de a observa modul de functionare al acestor programe, cat si de a evidentia cat de mult pot usura procesul de rezolvare al unor probleme relativ dificile fata de rezolvarea pe hartie. De asemenea, vom observa ca unele puzzle-uri sau probleme logice care par triviale pentru majoritatea oamenilor, pot crea dificultati programului, si vice versa.

2.2 Among Us

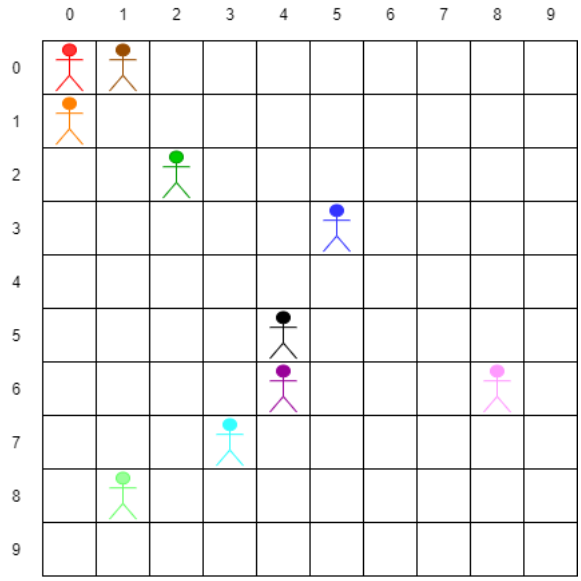
Among Us

Among Us este un joc video programat și lansat de către InnerSloth în anul 2018. Jocul constă în a găsi toți ucigașii, numiți și impostori. Scopul unui coechipier, numit și Crewmate este de a completa anumite misiuni precum repararea unor panouri electrice, pornirea reactorului, transmiterea datelor și altele. Avand in vedere ca este un joc bazat pe logica deductiva si mister, vom demonstra cum prover9 si mace4 ne pot ajuta sa aflam care sunt impostorii.

Pot fi în total zece jucători cu pana la 3 impostori. Obiectivul impostorilor este de a elimina toti membrii echipajului inainte de a fi demascati de catre acestia sau inainte sa fie completate toate task-urile.
Fiecare jucator are cate o culoare unica pentru a fi cat mai usor de diferentiat unul fata de celalalt.

Pe parcursul jocului vor exista intalniri de urgenta cu scopul de a determina suspectii in urma descoperirii unui cadavru sau la cererea unui jucator. Jocul se termina cand toti impostorii au fost demascati.

Pentru a putea simula un meci am dat fiecarui player o locatie si o culoare si am considerat urmatoarea conditie pentru a afla cine este impostor: daca un player este pe acceasi pozitie cu un cadavru si este considerat suspect de catre ceilalti jucatori, atunci este un impostor. Scopul problemei este de a verifica spusele jucatorilor si de a determina pe baza acestora si a knowlegde base-ului care sunt impostorii, sau de a genera toti impostorii posibili. Implementarea in Prover9 este aici. De asemenea, am considerat dialogurile dintre jucatori astfel:



EMERGENCY MEETING 1
1)color(red) <-> (sus(2,2) | sus(3,3)) & body(2,2). = Jucatorul rosu spune ca jucatorul din camera (2,2) sau (3,3) este suspect si este un cadavru in camera(2,2).
2)color(black) <-> -(exists x (sus(x,0))). Jucatorul black sustine ca nu exista un jucator care

sa fie suspect in camerele de pe linia 0.
EMERGENCY MEETING 2
3)color(purple) <-> exists x ((sus(x,0)) -> sus(6,x)). = Jucatorul mov spune ca daca este un suspect pe linia 0, este un suspect si pe linia 6.
4)color(cyan) <-> -sus(3,3) . = Jucatorul cyan spune ca jucatorul din camera (3,3) nu este suspect.
5)color(blue) <-> exists x (sus(x,x)). -> Jucatorul albastru spune ca nu exista un jucator suspect pe diagonala principala.
EMERGENCY MEETING 4
6)color(green) <-> sus(8,6). = Jucatorul verde spune ca jucatorul din camera (8,6) este suspect.
7)color(red) <-> -sus(0,0) & -sus(1,0). Jucatorul rosu spune ca nu exista un jucator suspect in camera (0,0) si nici in camera (1,0).
EMERGENCY MEETING 5
8)color(lime) <-> exists x ((sus(x,5)) -> sus(4,x)). = Jucatorul verde deschis spune ca daca exista suspect pe linia 5, exista suspect si pe coloana 4.
9)color(cyan) <-> sus(9,9) | sus(9,8) | body(0,0). = Jucatorul cyan spune ca jucatorul din camera (9,9) sau (9,8) este suspect sau este un cadavru in camera (0,0).
10)color(orange) <-> body(4,5). = Jucatorul portocaliu spune ca este un cadavru in camera (4,5).
EMERGENCY MEETING 6
11)color(purple) <-> body(4,5) & body(4,6) & sus(4,5). Jucatorul mov spune ca exista doua cadavre in camera (4,5) respectiv in (4,6) si este un jucator suspect in camera(4,5).
12)color(brown) <-> sus(4,5) | sus(4,6). = Jucatorul maro spune ca jucatorul din camera (4,5) sau (4,6) este suspect.
EMERGENCY MEETING 7
13)color(red) <-> sus(4,6). = Jucatorul rosu spune ca jucatorul din camera (4,6) este suspect.
14)color(cyan) <-> sus(4,6) | -sus(4,6). = Jucatorul din camera cyan spune ca jucatorul din camera (4,6) este sau nu este suspect.
15)color(lime) <-> exists x (body(3,x) -> body(6,3)). = Jucatorul din camera verde deschis spune ca daca exista un cadavru pe coloana 3, exista un cadavru si in camera(6,3).
16)color(pink) <-> body(4,6) & sus(4,6). = Jucatorul roz spune ca exista cadavru in camera(4,6) si un jucator suspect in camera(4,6).

Pentru a putea lua in considerare aceste dialoguri intre jucatori am definit si conditiile ca exista cel putin un impostor, fiecare jucator are o culoare cat si pozitiile acestora. Astfel pe baza dialogurilor si a conditiilor Prover9 poate demonstra ca presupunerea noastra de la sfarsit cum ca negru, verde si mov sunt impostori este adevarata.

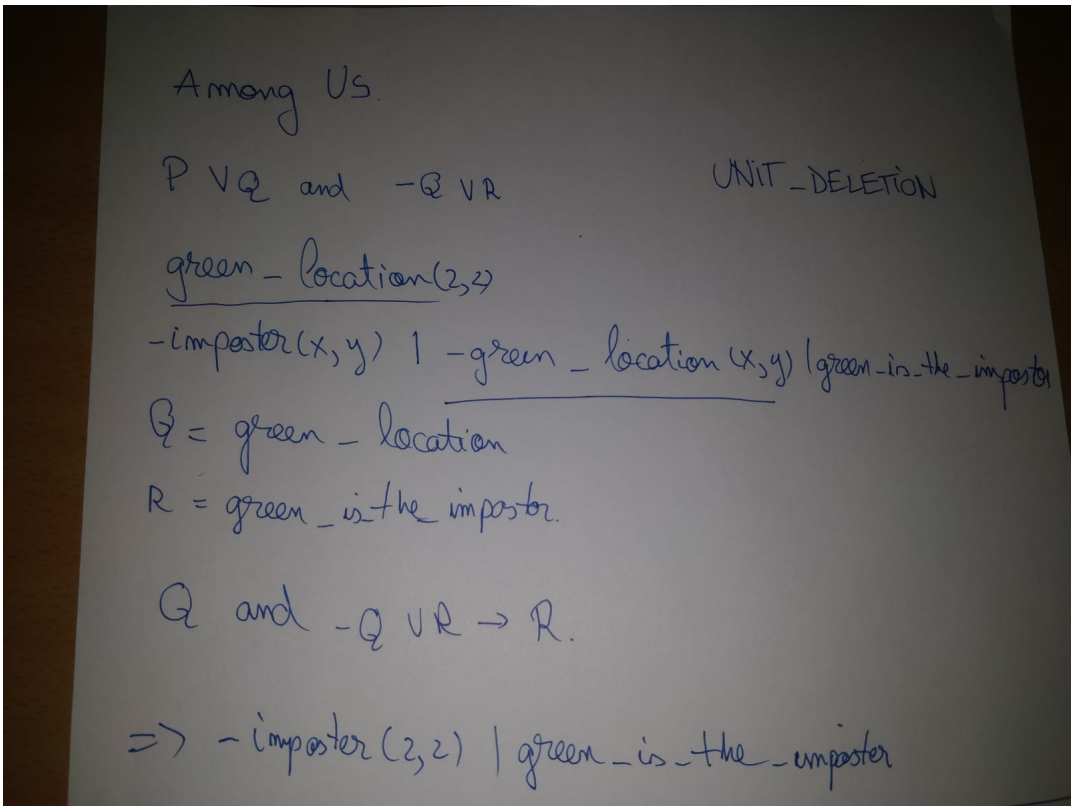
Acest lucru il putem confirma si noi pe baza linilor de dialog, de unde observam ca pozitiile cadavrelor,pozitiile jucatorilor respectivi si care din acestia sunt considerati suspecti. Pentru negru se poate observa Din dialogul 1 se poate observa ca exista un suspect in camera(2,2) sau in camera (3,3), dar din dialogul 4 se evidentiaza clar ca in camera(3,3) nu exista suspect. Din pozitiile jucatorilor observam ca jucatorul verde se afla in camera(2,2) si este primul impostor deoarece valideaza si celelalte doua conditii(sa fie suspect in camera lui, cadavru in camera lui).

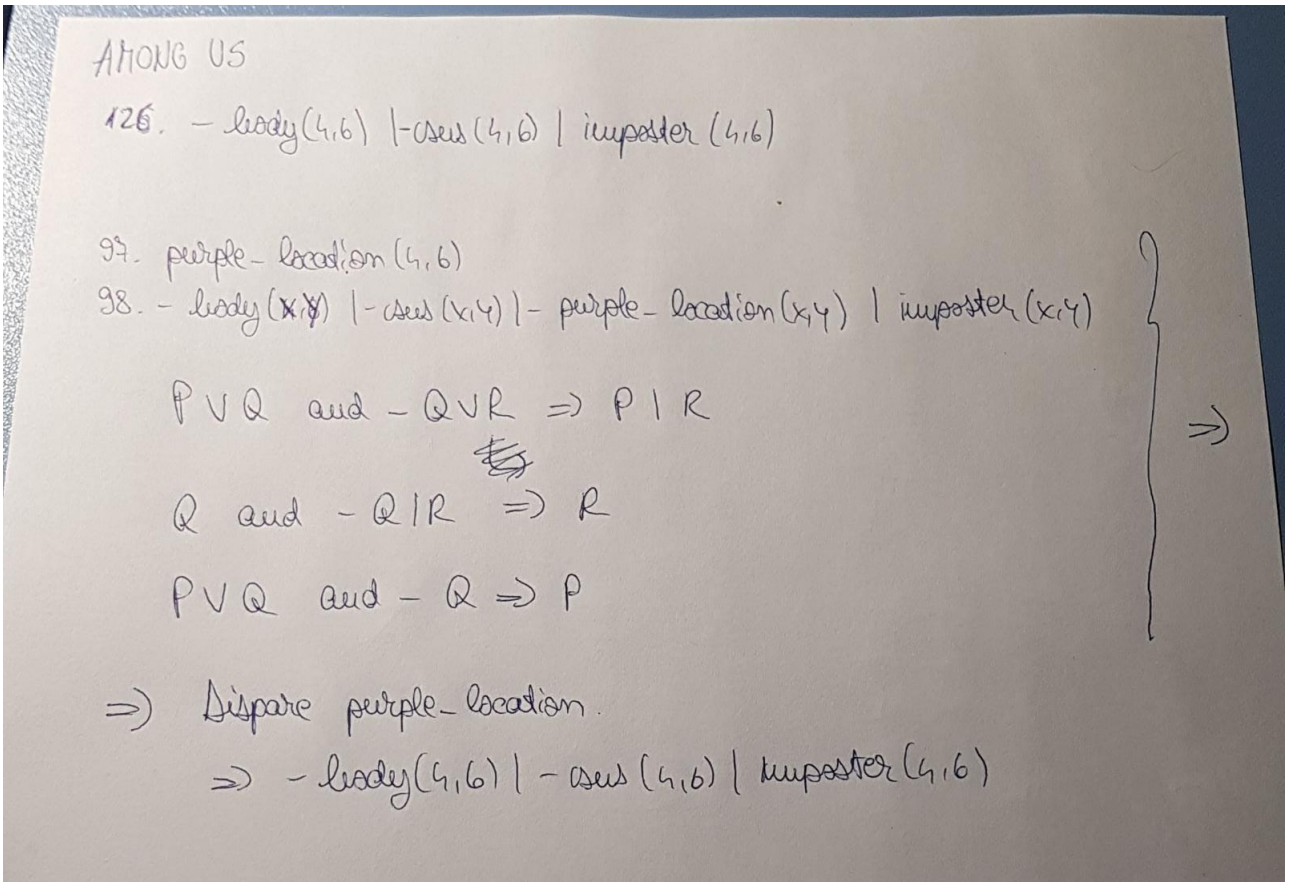
Din dialogul 11 jucatorul mov spune ca a descoperit doua cadavre in camerele (4,5) si (4,6), dar ne mai spune ca este un suspect in camera(4,5). Astfel jucatorul mov prinde al doilea impostor si anume pe colegul sau verde deoarece pozitia sa corespunde cadavrului si suspectului.(Jucatorul mov pare incepator si si-a demascat colegul impostor)

Jocul se termina la urmatoarea intalnire de urgenta(teorema va fi demonstrata) prin indiciul declarat de jucatorul rosu si anume ca ultimul suspect se afla in camera(4,6). Ultima

conditie fiind verificata inseamna ca al treilea si ultimul impostor, mov, a fost demascat.

Pe tot parcursul jocului impostorii au incercat sa induca in eroare pe ceilalti jucatori folosind indicii false pentru a-si dovedi nevinovatia.





black_is_the_imposter: 1
blue_is_the_imposter: 0
brown_is_the_imposter: 0
cyan_is_the_imposter: 0
green_is_the_imposter: 1
lime_is_the_imposter: 0
orange_is_the_imposter: 0
pink_is_the_imposter: 0
purple_is_the_imposter: 1
red_is_the_imposter: 0

body:	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	1	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

imposter:	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	1	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

sus:	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	1	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	0	0
9	0	0	0	0	0	0	0	0	0	1

Table 2.1

2.3 Superheroes puzzle

Superheroes Puzzle Grid Logic

Puzzle-ul de tip Grid Logic este un gen de puzzle-uri logice in care jucatorul primeste un tabel, un scenariu, obiectele de dedus si anumite indicii. Grila va afișa un anumit număr de categorii (în acest caz, 4) și un anumit număr de obiecte pe categorie (în acest caz, 4). Fiecare obiect se potrivește cu un singur alt articol din fiecare categorie și niciun obiect dintr-o categorie nu va fi asociat vreodată cu același obiect din altă categorie. Jucatorul trebuie sa foloceasca indiciile furnizate si unele constrangeri de domeniu pentru a afla solutia acestuia si a completa tabelul.
Grid logic-ul pe care l-am ales se numeste Superheroes Origins.Mai jos puteti observa o imagine care prezinta obiectele si categoriile acestuia cat si indiciile oferite jucatorului pentru a completa acest puzzle. Implementarea in Mace4 este aici.

		AKA				Appearance				Origin			
		Laufeyson	Matthew	Petragon	William	1962	1963	1964	1965	Giant	Hybrid	Mutant	Superhuman
Hero	Daredevil												
	Gorgon												
	Loki												
	Sandman												
Origin	Giant												
	Hybrid												
	Mutant												
	Superhuman												
Appearance	1962												
	1963												
	1964												
	1965												

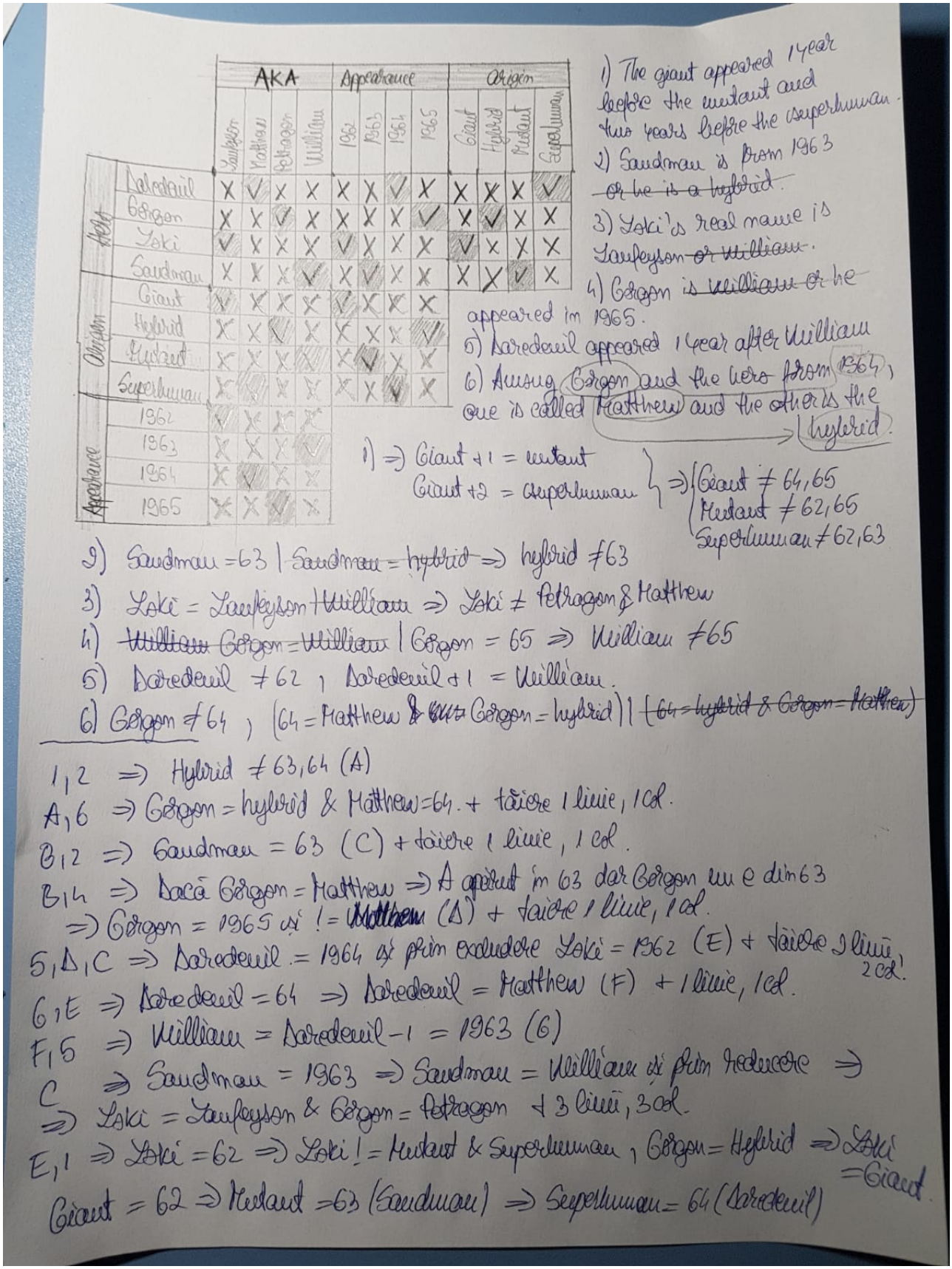
Indicii:

- Gigantul a apărut cu 1 an înainte de mutant și cu doi ani înainte de superhuman.
- Sandman este din 1963 sau este un hibrid.

- Numele real al lui Loki este Laufeyson sau William.
- Gorgon este William sau a apărut în 1965.
- Daredevil a apărut la 1 an după William.
- Dintre Gorgon și eroul din 1964, unul se numește Matthew, iar celălalt este hibridul.

Solutie problema:

- Din indiciul 1 aflam ca Giant nu poate fi din anul 1964 si 1965,Mutant nu poate fi din anul 1962 si 1965, si SuperHuman nu poate fi din anul 1962 si 1963.
- Din indiciul 2 aflam ca Hybrid nu poate fi din anul 1963 si Sandman poate fi din anul 1963 sau este hybrid.
- Din indiciul 3 aflam ca numele lui Loki nu poate fi Petragon,dar nici Matthew.
- Din indiciul 4 aflam ca William nu poate fi din anul 1965 si mai aflam ca numele lui Gorgon este Willian sau acesta poate fi din anul 1965.
- Din indiciul 5 aflam ca Daredevil nu poate fi din anul 1962.
- Din indiciul 6 aflam ca Gorgon nu poate fi din anul 1964 si mai aflam ca personajul din anul 1964 are numele Matthew sau originea hybrid SAU personajul Gorgon are nuemele Mathew sau are originea hybrid.
- Din primele 2 indicii aflam ca originea Hybrid nu poate sa fie din anul 1963 sau 1964. (A)
- Din indiciile A si 6 putem observa ca Gorgon este de originea Hybrid si supereroul cu numele Matthew este din anul 1964. (B)
- Din indiciile B si 2 putem deduce ca personajul Sandamn a aparut in anul 1963. (C)
- Din indiciile B si 4 putem presupune ca Gorgon are numele Matthew ceea ce ar insemna ca a aparut in 1963 dar Gorgon nu a aparut in 1963, ceea ce inseamna ca Gorgon este din anul 1965 si nu are numele Matthew.(D)
- Din indiciile 5,D si C observam ca singurul an disponibil al supereroului Daredevil este 1964. Prin regula excluderii putem spune ca si Loki este din anul 1962. (E)
- Din indiciile 6 si E stim ca Daredevil este din 1964 si stim ca numele Matthew este tot din acelasi an ceea ce rezulta ca numele lui Daredevil este Matthew. (F)
- Din indiciile F si 5 stim ca William este cu un an mai devreme decat Daredevil si Daredevil este 1964 ceea ce rezulta ca numele William este din anul 1963.
- Din indiciul C stim ca Sandman este din anul 1963 ceea ce inseamna ca numele sau este William, iar prin excludere celelalte nume vor fi Loki = Laufeyson si Gorgon = Petragon.
- Din indiciile E si 1 stim ca anul lui Loki este 1962 ceea ce rezulta ca acesta nu poate fi Mutant,SuperHuman sau Hybrid.
- Din indiciul 1 aflam ca Giant este din anul 1962, Mutant 1963 si Superhuman 1964.



1) The giant appeared 1 year before the mutant and two years before the superhuman.
2) Saudman is from 1963 ~~or he is a hybrid~~.
3) Loki's real name is Laufeyson ~~or William~~.
4) Gorgon ~~is William or~~ he appeared in 1965.
5) Doredevil appeared 1 year after William
6) Among Gorgon and the hero from 1964, one is called Matthew and the other is the hybrid.

1) $\Rightarrow$ Giant + 1 = mutant
Giant + 2 = superhuman
$\Rightarrow$ { Giant $\neq$ 64, 65
Mutant $\neq$ 62, 65
Superhuman $\neq$ 62, 63

2) Saudman = 63 | Saudman = hybrid $\Rightarrow$ hybrid $\neq$ 63

3) Loki = Laufeyson + William $\Rightarrow$ Loki $\neq$ Petragon & Matthew

4) ~~William Gorgon = William~~ | Gorgon = 65 $\Rightarrow$ William $\neq$ 65

5) Doredevil $\neq$ 62, Doredevil + 1 = William.

6) Gorgon $\neq$ 64, (64 = Matthew & Gorgon = hybrid) || ~~(64 = hybrid & Gorgon = Matthew)~~

1,2 $\Rightarrow$ Hybrid $\neq$ 63, 64 (A)
A, 6 $\Rightarrow$ Gorgon = hybrid & Matthew = 64. + faicre 1 linie, 1 col.
B, 2 $\Rightarrow$ Saudman = 63 (C) + faicre 1 linie, 1 col.
B, 4 $\Rightarrow$ Daca Gorgon = Matthew $\Rightarrow$ A apărut în 63 dar Gorgon nu e din 63
$\Rightarrow$ Gorgon = 1965 așa ! = ~~Matthew~~ (D) + faicre 1 linie, 1 col.
5, D, C $\Rightarrow$ Doredevil = 1964 așa prin excludere Loki = 1962 (E) + faicre 1 linie, 2 col!
6, E $\Rightarrow$ Doredevil = 64 $\Rightarrow$ Doredevil = Matthew (F) + 1 linie, 1 col.
F, 5 $\Rightarrow$ William = Doredevil - 1 = 1963 (G)
C $\Rightarrow$ Saudman = 1963 $\Rightarrow$ Saudman = William așa prin reducere $\Rightarrow$
$\Rightarrow$ Loki = Laufeyson & Gorgon = Petragon + 3 linii, 3 col.
E, 1 $\Rightarrow$ Loki = 62 $\Rightarrow$ Loki ! = Mutant & Superhuman, Gorgon = Hybrid $\Rightarrow$ Loki = Giant.
Giant = 62 $\Rightarrow$ Mutant = 63 (Saudman) $\Rightarrow$ Superhuman = 64 (Doredevil)

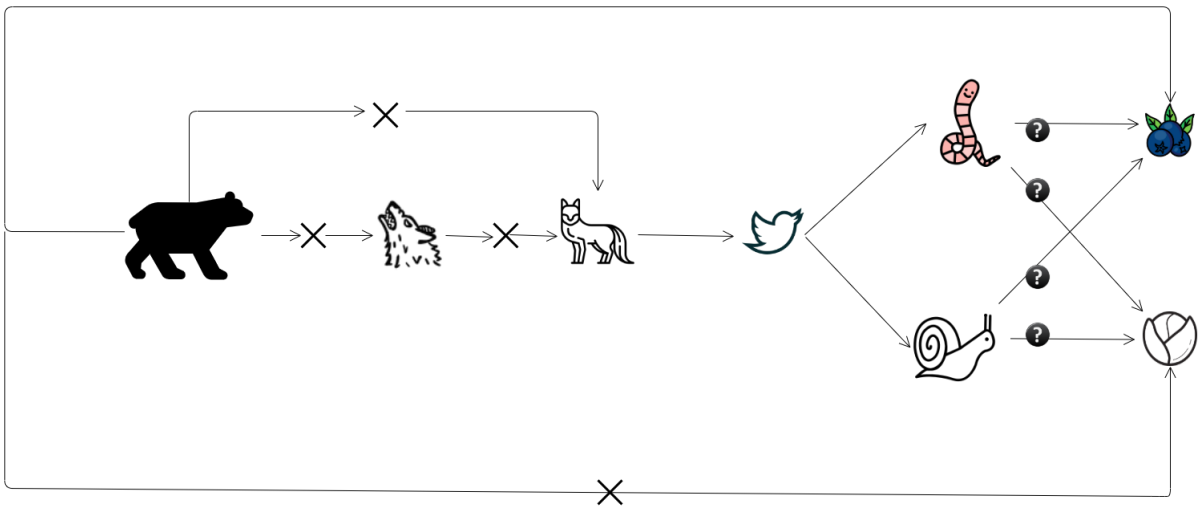
2.4 Utilajul lui Schubert

Utilajul lui Schubert

Avem o padure unde avem urmatoarele animale si plante:

- Ursii, lupii, vulpile, pasarile, gandacii si melcii sunt animale si exista cel putin un specimen pentru fiecare.
- Exista si varza si afine, acestea fiind plante.
- Fiecare animal mananca plante sau toate celelalte animale care nu sunt mai marisi mananca plante.
- Ursii sunt mai mari decat lupii, care sunt mai mari decat vulpile, care sunt maimari decat pasarile, care sunt mai mari decat gandacii si melcii.
- Ursii nu mananca lupi, lupii nu mananca vulpi sau varza, pasarile mananca gandacidar nu si melci.
- Ursii mananca afine iar gandacii si melcii mananca unele plante.

Trebuie sa demonstram ca exista in padure un animal care mananca toate animalele care mananca varza.



Solutia acestei probleme este relativ usoara si poate fi gasita foarte rapid. Se observa ca ursii si lupii nu respecta regula necesara pentru rezultatul dorit, deci mai ramanem doar cu vulpile si pasarile, dar deoarece pasarile sunt mai mici decat vulpile, acestea raman singurele care verifica conditia dorita.

O asemenea problema care pare relativ banala pentru un om, va avea nevoie de o descriere mai laborioasa pentru a putea fi demonstrata de catre prover. Astfel, aceasta problema este un exemplu perfect pentru evidentierea faptului ca o asemenea problema triviala ne poate da batai de cap daca dorim sa o transpunem intr-un demonstrator de teoreme. Implementarea in Prover9 este aici.

SCHUBERT'S PROBLEM

P ∪ Q and ¬Q ∪ R UNIT_DELETION

47. ∃x (C3)

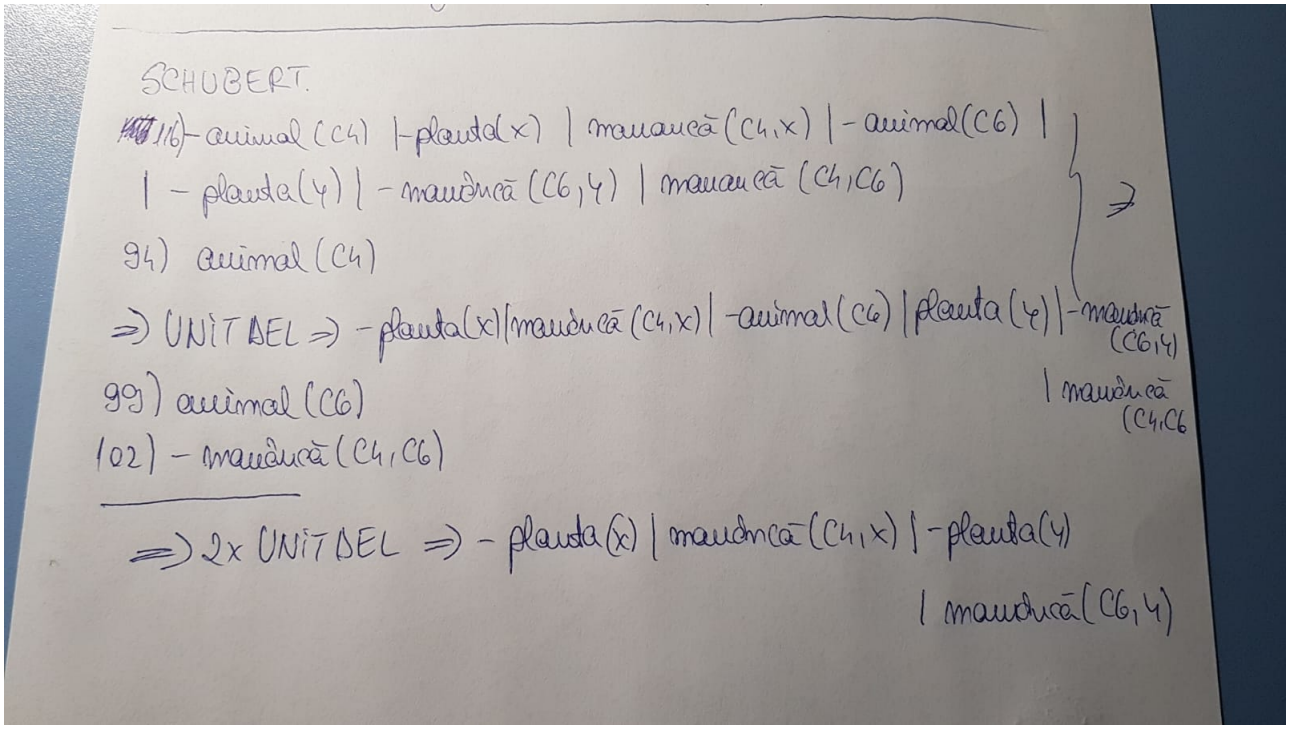
49. ¬∃x(x) | ¬Pasare(y) | mai_mare(x,y)

Q = ∃x(C3)

R = mai_mare(x,y)

Q and ¬Q ∪ R → R

=> ¬Pasare(y) | mai_mare(x,y)



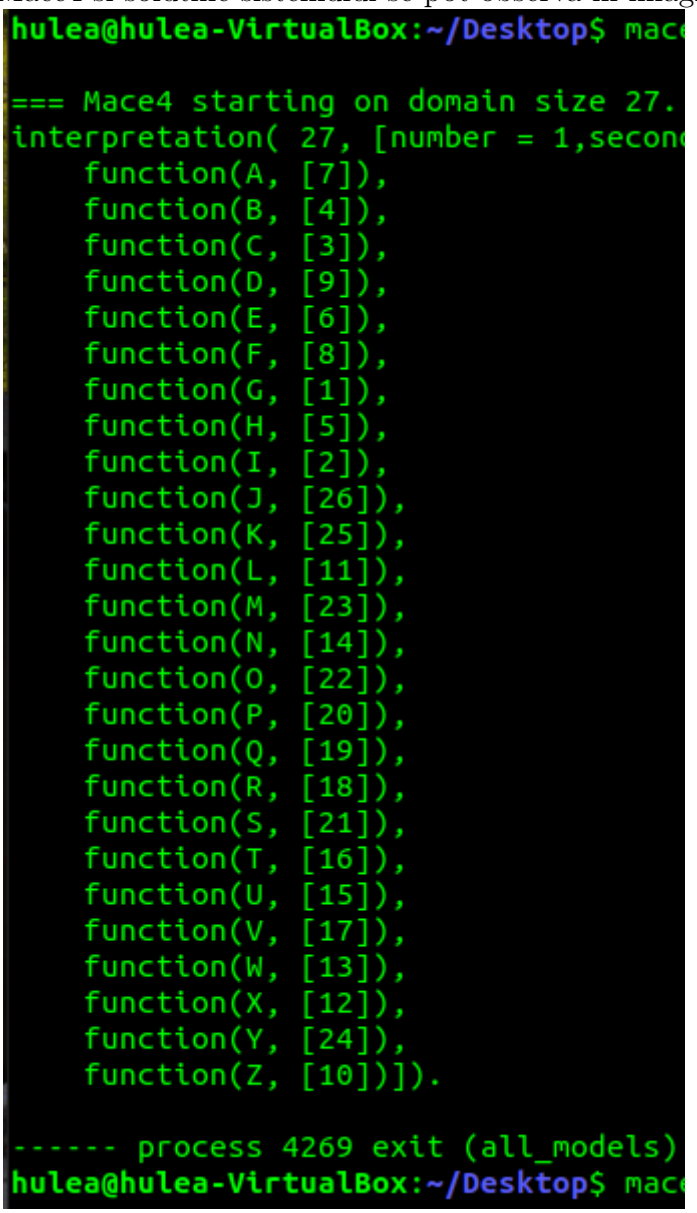
2.5 Alphabetical Logic Equations

Logic Equations Puzzle

Logic Equations Puzzle este un puzzle matematic in care jucatorul trebuie să afle valorile variabilelor rezolvând ecuațiile logice folosind o grila NxN, unde N este numarul de variabile pe care le detine sistemul matematic.Variabilele reprezintă numere întregi unice variind de la 1 la numărul de variabile. Scopul implementarii acestui puzzle este de a evidentia cum o problema relativ simpla poate da mari batai de cap programului. Asadar, daca avem un numar mic de variabile, rezultatul va fi afisat dupa rulare la cateva secunde, dar daca numarul acestora este mai mare, timpul va creste exponential. Spre exemplu, implementarea. sistemului de ecuatii de mai jos in mace4 a afisat rezultatul in nu mai putin de o ora.

$$
\begin{cases}
C * G = C. \\
6 * C = C * E. \\
B + 3! = 2 * A. \\
3 * I >= B. \\
F! = A + D. \\
F = C + H. \\
3 * I <= 9. \\
C + E >= A. \\
A + F + H + 1 = C + D * I. \\
E! = H + I. \\
H > B. \\
J > K. \\
K = 25. \\
2 * L = O. \\
M = Y + -1. \\
N + -1 = Z + C. \\
O + 2 = Y. \\
P = 2 * Z. \\
Q = L + F. \\
10 * R = 10 * V + Z. \\
3 * S = 3 * P + G. \\
T = U + 1. \\
U = 3 * H. \\
V = T + 1. \\
2 * W = J. \\
X = B + 2 * C + 2. \\
Y > M \& Y < K. \\
Z = A + G + 2.
\end{cases}
$$

Rezultatele din Mace4 si solutiile sistemului se pot observa in imaginea de mai jos:



2.6 Who is the spy?

Metapuzzle: Who is the spy?

Un metapuzzle este un puzzle despre puzzle-uri. Ne este dat un puzzle care nu are suficiente date pentru a fi rezilvat, dupa care ne este spus daca cineva ar putea sau nu ar putea rezolva puzzle-ul cu ajutorul unor informatii aditionale. Desigur, nu intotdeauna ne este spus care este aceasta informatie, dar putem primi informatii partiale despre aceasta, astfel avand posibilitatea de a rezolva puzzle-ul.

Enuntul problemei este urmatorul:
Un caz la tribunal implica 3 inculpati: A, B si C. Se stie din prealabil ca unul dintre ei spune doar adevarul(adica este knight), unul minte intotdeauna(adica este knave) si celalalt este un spion, care este normal(adica poate sa minta sau poate sa spuna adevarul). Scopul acestui proces este de a afla care dintre cei trei este spionul.

Prima oara, lui A i s-a spus sa faca o afirmatie. Acesta a spus ca C ori este knave, ori este spion, dar nu este spus care. Urmatorul a fost B care a spus ca A ori este knight, ori este knave, ori este normal, dar la fel ca la A, nu ne este spus explicit ce este A. Ultimul a fost C care a spus ca B ori este knight, ori este knave, ori este normal. Dupa acestea, judecatorul si-a dat seama care dintre ei este spionul si l-a condamnat la inchisoare.

Acest caz a fost descris si unui logician, care a lucrat asupra cazului o perioada de timp dar acesta nu a stiut raspunsul. Dupa ce i-a fost spus ce a zis A, acesta si-a dat seama care era spionul.

Care era spionul - A,B,C?

Solutia acestei probleme se poate afla doar prin presupuneri. Avand 3 acuzati care fiecare poate fi de cate un fel diferit, rezolvarea problemei devinde destul de lunga daca consideram toate cazurile posibile.
In primul rand, incepem prin a bifurca posibilitatile prin presupunerea ca logicianului i-a fost

spus ca A a afirmat ca C este knave. Astfel avem: Posibilitatea 1: A a spus ca C este knave. Acum vom avea alte 3 cazuri posibile in functie de ce a spus B, cazuri care trebuie analizate. Cazul 1: B a spus ca A este knight. Atunci, daca A este knight, C este knave deoarece A a spus ca C este knave(A fiind knight, nu poate minti), asadar B fiind spionul. Daca A este knave, atunci afirmatia lui B este falsa, ceea ce inseamna ca B este spionul si C este knave, Ultimul caz este daca A ar fi spionul, atunci afirmatia lui B ar fi falsa, deci B ar fi knave si C ar fi knight. Asadar avem urmatoarele cazuri:
(1)A knight, B spy, C knave
(2)A knave, B spy, C knight
(3)A spy, B knave, C knight
Acum presupunem ca C a spus ca B este spionul. Atunci (1) si (3) nu mai sunt valabile deoarece daca, (1), C, un knave, nu ar putea afirma ca B este spion, pentru ca B este spion, si (3), C, un knight, nu poate afirma ca B este spion pentru ca B nu este. Asadar mai ramane doar optiunea (2), unde judecatorul ar stii ca B este spionul.
Preuspunem ca C a spus ca B este knight. Atunci (1) este singura posibilitate, si judecatorul

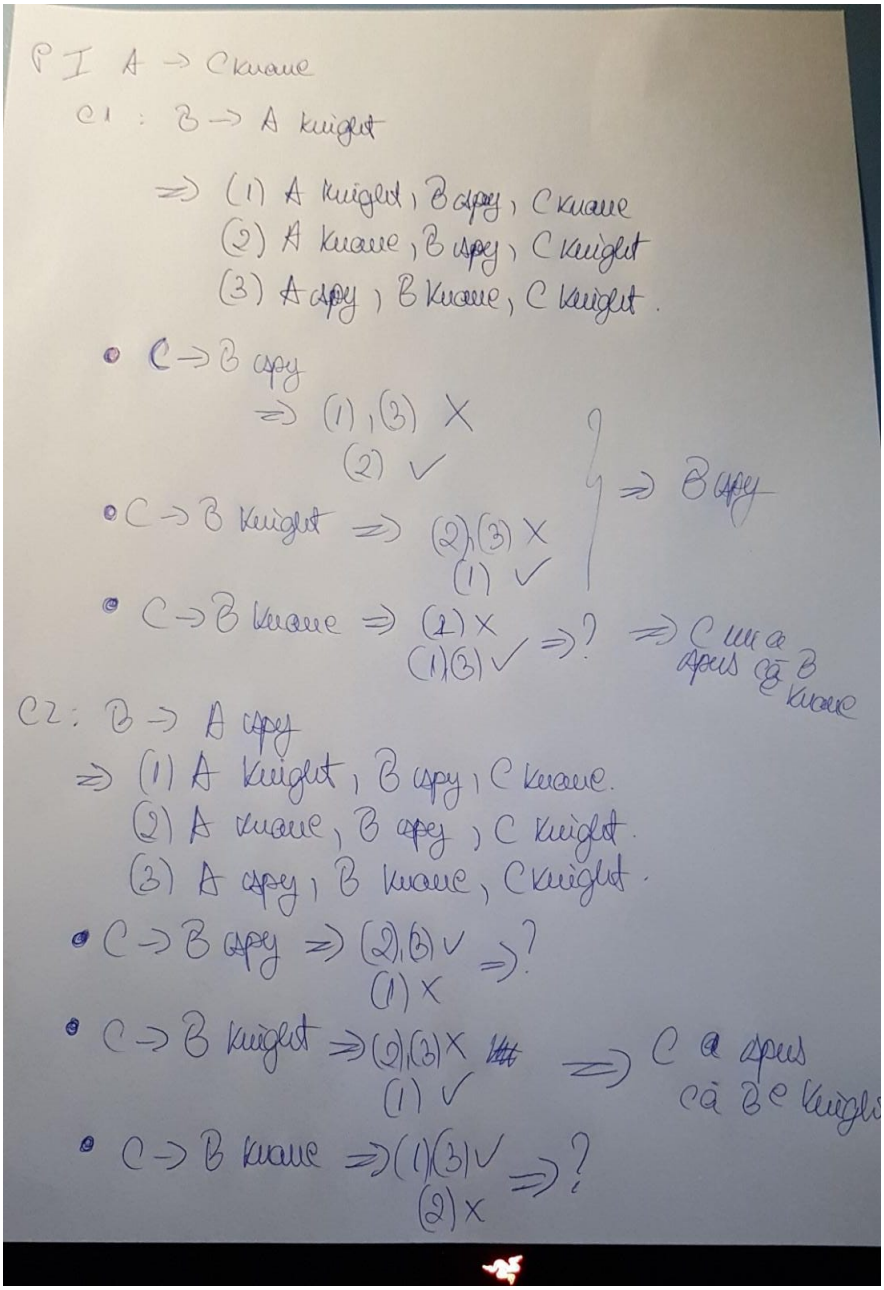
ar stii din nou ca B este spionul.
Presupunem ca C a zis ca B este knave. Atunci judecatorul nu ar fi putut sa stie care dintre cazurile (1) si (3) este adevarat, deci nu ar fi putut alege spionul intre A si B. Asadar, C nu a spus ca B este knave.
Daca cazul este adevarat, atunci B este singurul care poate fi acuzat.
Cazul 2: B a spus ca A este spion. La fel ca mai sus, avem urmatoarele cazuri:
(1)A knight, B spy, C knave
(2)A knave, B spy, C knight
(3)A spy, B knight, C knave
Daca C ar fi spus ca B este spion, atunci (2) si (3) ar fi ambele adevarate si judecatorul nu s-ar putea decide. Daca C a spus ca B este knight, atunci doar (1) e adevarata, si judecatorul l-ar acuza pe B. Daca C a spus ca B este knave, atunci si (1) si (3) sunt adevarate. Asadar, C a spus ca B este knight, si B a fost acuzat si la Cazul 2.
Cazul 3: B a spus ca A este knave. In acest caz avem 4 posibilitati:
(1)A knight, B spy, C knave
(2)A knave, B spy, C knight
(3)A knave, B knight, C spy
(4)A spy, B knave, C knight
Daca C ar fi spus ca B este spion, (2) si (3) ar fi adevarate. Daca C ar fi spus ca B este knight, (1) si (3) ar fi adevarate. Daca C ar fi spus ca B este knave, atunci (1),(3) si (4) ar fi adevarate, si judecatorul nu s-ar putea decide care e spionul in niciunul dintre cazuri.
Avand in vedere ca pentru Cazul 3 nu avem solutii, stim sigur ca unul dintre Cazul 1 si 2 sunt adevarate, in ambele B fiind acuzat.
Posibilitatea 2: Presupunem ca A a spus ca C este spion. Exista o posibilitate in care judecatorul l-ar fi putut condamna pe A: Presupunem ca B a spus ca A este knight si C a spus ca B este knave. Daca A este spionul, B poate fi knave(care il acuza fals pe A ca fiind knight), si C este knight. Deci, A(spionul) il acuza fals pe C ca fiind spion si este posibil ca A,B si C sa faca aceste afirmatii si ca A sa fie spion. Daca B ar fi spion , A ar fi knave si ca sa il acuze pe C ca fiind spion, C ar trebui sa fie knave deoarece a spus ca B este knave, ceea ce nu e posibil. Daca C ar fi spion, atunci A ar fi knight pentru ca a spus adevarul, C ar fi spion si B ar trebui sa fie tot knight pentru ca a spus adevarul, adica ca A este knight, ceea ce nu e posibil. Asadar, este posibil ca si A sa fie spion.
Modul prin care B ar putea fi acuzat este urmatorul: presupunem ca B a spus ca A este knight si C a spus ca B este spion. Daca A este spion, B este knave pentr ca spune ca A este knight si C este tot knave pentru ca spune ca B este spy, ceea ce nu e posibil. Daca C este spion, atunci A este knight, si B este tot knight pentru ca spune ca A este knight, ceea ce nu e posibil. Dar daca B este spion, atunci nu avem contradictie(A poate fi knave ce spune ca C este spion, C poate fi knight care spune ca B e spion si B poate spne ca A este knight). Deci, este posibil ca A,B si C sa faca aceste afirmatii si B sa fie acuzat.
In concluzie, exista o posibilitate ca A sa fie acuzat si o posibilitate ca B sa fie acuzat. Asadar, daca logicianului i-a fost spus ca A a zis ca C este spionul, atunci nu exista nici o cale ca logicianul sa fi putut rezolva problema. Daca ne este spus ca logicianul nu a putut rezolva problema, putem deduce ca A a spus ca C este knave, si dupa cum am vazut si mai sus, judecatorul l-ar fi putut condamna doar pe B. Asadar, B este spionul.

Aceasta problema este foarte greu de demonstrat deoarece trebuie luate toate cazurile posibile in considerare, dar rezolvarea sa in mace4 este relativ simpla si usor de inteles.

Rezultatul din mace4 este corect conform demonstatiei de mai sus

a1:	0
a2:	1
a3:	0
b1:	0
b2:	0
b3:	1
c1:	1
c2:	0
c3:	0
da:	0
db:	1
dc:	1

Table 2.2



2.7 Knight, Knave and Normals

Knight,Knave and Normals - Puzzle 46

Acest puzzle este o problema care se refera la trei tipuri de oameni: Knights, care vor spune intotdeauna adevarul, Knaves care vor minti mereu si Normal care pot spune si adevaruri si minciuna. Insula Bahava este o insula feminista, prin urmare si femeile sunt numite tot Knights,Knaves,Normals. O imparateasa a acestei instule a adoptat un drecret, potrivit caruia un Knight se poate casatori doar cu un Knave si vice-versa. (Persoanele de tip Normal se vor putea casatori doar cu altele de acelasi tip).
Enuntul problemei este urmatorul:
Pe insula Bahava exista doua cupluri: A si B. In decursul unui interviu trei din cele patru persoane au acordat urmatoarea marturie: primul, sotul din cuplul A, a afirmat ca sotul din cuplul B, este un Knight. Al doilea,sotia din cuplul A, a aprobat ce a spus sotul si anume ca sotul din cuplul B este un Knight. Ultima marturie a fost acordata de catre sotia din cuplul B care afirma ca sotul ei este un Knight.
In urma implementarii in mace4, programul a generat o solutie intr-un timp mult mai rapid in comparatie cu incercarile mai multor logicieni de a rezolva aceasta problema.

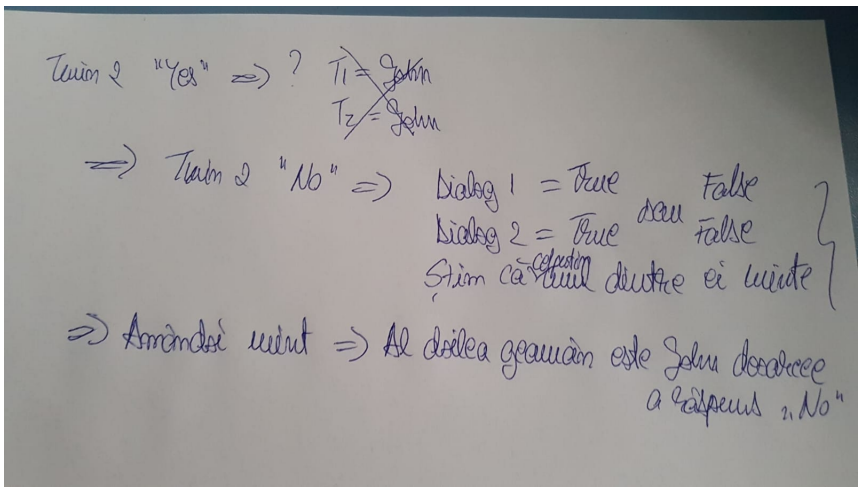
Solutia problemei 46 este ca toti 4 sunt normali si toate cele 3 afirmatii sunt minciuni. In primul rand, Doamna B trebuie sa fie normala deoarece daca ar fi fost knight, sotul ei ar fi fost knave, deci ea nu ar fi mintit spunand ca el este knight. Daca ea ar fi fost knave, sotul ei ar fi fost knight, dar atunci ea nu ar mai fi zis adevarul. In concluzie, Doamna B este normala, impreuna cu Domnul B. Asta inseamna ca domnul si doamna A au mintit amandoi. Deci, niciunul nu este knight, si nu poti fi amandoi knaves, deci sunt ambii normali. Rezultatele din Mace4 si solutiile problemei se pot observa in imaginea de mai jos:

```
=== Mace4 starting on domain size 4. ===
interpretation( 4, [number = 1,seconds = 0],
    function(b1, [0]),
    function(b2, [0]),
    function(f1, [0]),
    function(f2, [0]),
    relation(knave(_), [0,0,0,0]),
    relation(knight(_), [0,0,0,0]),
    relation(normal(_), [1,1,1,1]),
    relation(true(_), [0,0,0,0]),
    relation(Different(_,_), [
        1,0,0,0,
        0,0,0,0,
        0,0,0,0,
        0,0,0,0]),
    relation(married(_,_), [
        1,0,0,0,
        0,0,0,0,
        0,0,0,0,
        0,0,0,0])]).
```

2.8 Alte metapuzzle-uri cu rezolvari mai scurte

1)Ancheta lui John
Acest caz este despre 2 gemeni identici. Se stie ca cel putin unul dintre ei nu spune niciodata adevarul, dar nu se stie care dintre ei. Unul dintre ei este numit John si a comis o crima(acest fapt nu implica ca John este cel care minte tot timpul). Scopul acestei investigaii este de a afla care este John. Dialogul celor 2 gemeni cu judecatorul este urmatorul:
Judecator,catre primul geaman:"Tu esti John?".
Primul geaman:"Da, eu sunt."
Judecator, catre al doilea geaman:"Tu esti John?"
Al doilea geaman a raspuns da sau nu, si judecatorul a stiut care este John.
2)Un metapuzzle din Transilvania
In Transilvania exista 4 tipuri de locuitori: oameni normali, oameni nebuni, vampiri normali, si vampiri nebuni. Oamenii si vampirii nebuni spun intotdeauna adevarul, oamenii nebuni si vampirii spun doar minciuni. Trei logicieni discutau despre vizitele lor in Transilvania, unde l-au intalnit pe Igor. Primul logician l-a intrebat pe Igor daca este om normal. Al doilea l-a intrebat daca este vampir normal. Al treilea l-a intrebat daca este vampir nebun. Igor a raspuns da sau nu la toate intrebarile. Ce tip de locuitor era Igor?
3)Knight-Knave Metapuzzle
Proprietile despre knight si knave se pastreaza si la aceasta problema. Un logician viziteaza o insula cu doi locuitori, A si B. Acesta il intreaba pe A, "Sunteti amandoi knight?" si A raspunde cu da sau nu. Logicianul se gandeste un timp, dupa care isi da seama ca nu are suficiente informatii pentru a le determina tiipul celor 2 si il intreaba pe A, "Sunteti amandoi de acelasi tip?". A raspunde cu da sau nu si logicianul isi da seama de tipurile lor. Care de ce tip era?
4)Knight-Knave-Normal Metapuzzle
Pe o insula, pe langa knights si knaves mai existau si oameni normali. Acestia puteau sa minta sau sa spuna adevarul. Un logician viziteaza aceasta insula si intalneste 2 locuitori A si B. El stia deja ca unul dintre acestia este knight si celalalt normal, dar nu stia exact care. Asadar, l-a intrebat pe A daca B este normal si acesta i-a raspuns cu da sau nu. Care dintre cei 2 este normal?

Rezolvarile logice ale problemelor sunt urmatoarele:
1)Daca al doilea geaman ar fi raspuns da, atunci judecatorul nu ar fi avut de unde sa stie care e John, deci putem presupune cu siguranta ca al doilea geaman a raspuns "Nu". Asta inseamna ca ambii gemeni mint sau spun adevarul, dar noi stim ca cel putin unul dintre ei minte. Asadar, daca ambii gemeni mint, putem deduce usor ca al doilea geaman este "John".

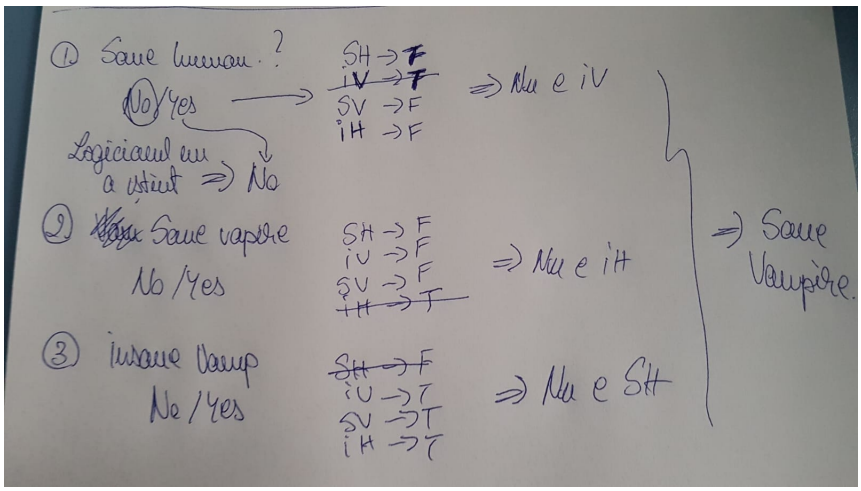



```
hulea@hulea-VirtualBox:~/Desktop$ mace4 -c -n 2 -m -1 -f john.in | interpformat

=== Mace4 starting on domain size 2. ===

------ process 2798 exit (all_models) ------
interpretation( 2, [number = 1,seconds = 0], [
    relation(d1, [0]),
    relation(d2, [1]),
    relation(j1, [0]),
    relation(j2, [1]),
    relation(t1, [0]),
    relation(t2, [1])]).
```

2)Daca Igor ar fi om normal,om nebun sau vampir normal el ar fi raspuns "da" la intrebarea primului logician. Ar fi raspuns nu daca era vampir nebun. Daca Igor ar fi raspuns nu, atunci logicianul ar fi stiut ca igor este vampir nebun, deci putem deduce din moment ce logicianul nu a stiut ce este igor, ca acesta a raspuns da, deci Igor nu este vampir nebun.
La a doua intrebare, doar un om nebun ar fi raspuns cu da, restul ar fi raspuns cu nu. La fel ca mai sus, putem asuma ca igor nu este om nebun.
Urmarind aceeasi logica, la a treia intrebare doar un om normal ar fi raspuns nu, celelalte 3 tipuri ar fi raspuns cu da. Asadar, putem asuma ca Igor nu e nici om normal. In concluzie, prin eliminare, rezulta ca Igor este un vampir normal.

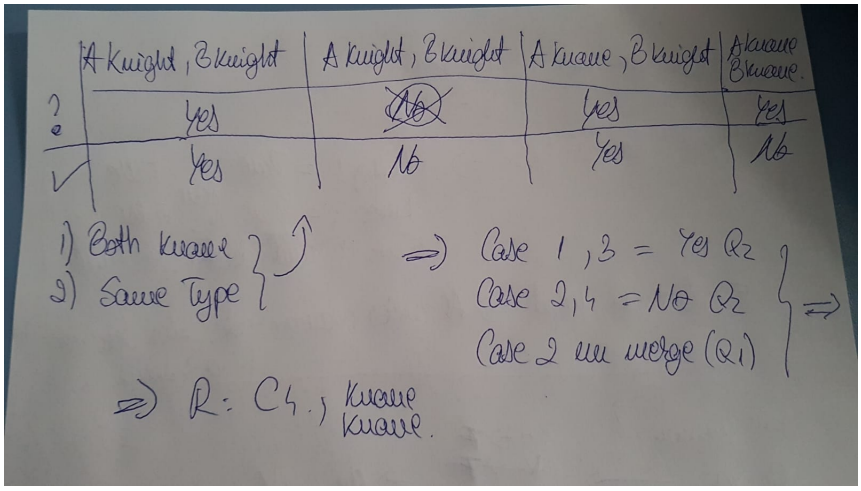



```
hulea@hulea-VirtualBox:~/Desktop$ mace4 -c -n 2 -m -1 -f igor.in | interpformat

=== Mace4 starting on domain size 2. ===

------ process 22783 exit (all_models) ------
interpretation( 2, [number = 1,seconds = 0], [
    relation(d1, [1]),
    relation(d2, [1]),
    relation(d3, [1]),
    relation(h1, [0]),
    relation(s1, [1])]).
hulea@hulea-VirtualBox:~/Desktop$
```

3)La aceasta problema avem 4 cazuri posibile:
-Ambii sunt knight
-A knight, B knave
-A knave, B knight
-Ambii sunt knave
La prima intrebare doar cazul 2 ar fi raspuns nu. Din moment ce acesta nu si-a dat seama de tipul celor 2, putem deduce ca raspunsul la prima intrebare a fost da. La a doua intrebare in cazul 1 si 3 raspunsul ar fi fost da, iar la cazul 2 si 4 raspunsul ar fi fost nu. Asadar daca raspunsul la a doua ar fi fost da, logicianul nu ar fi putut alege intre 1 si 3, iar daca ar fi fost nu, ar fi trebuit sa ghiceasca intre 2 si 4. Dar noi stim ca al doilea caz nu e bun datorita primei intrebari, deci putem deduce ca A si B sunt ambii knave, adica varianta 4.

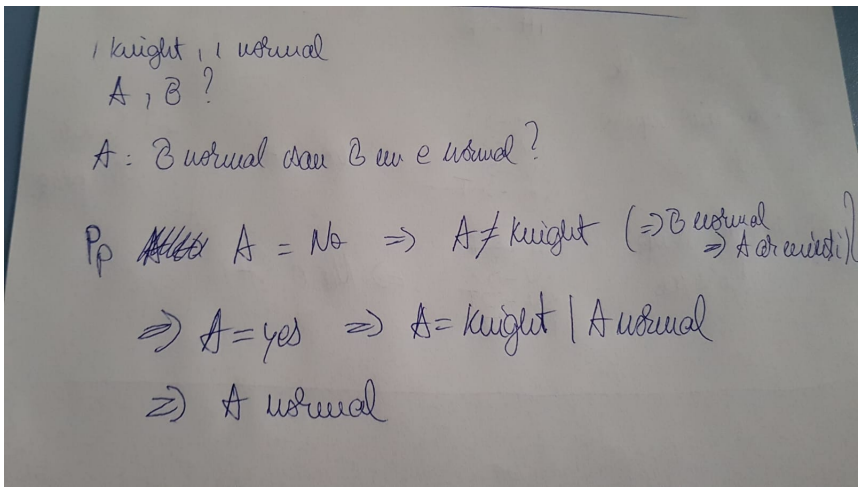


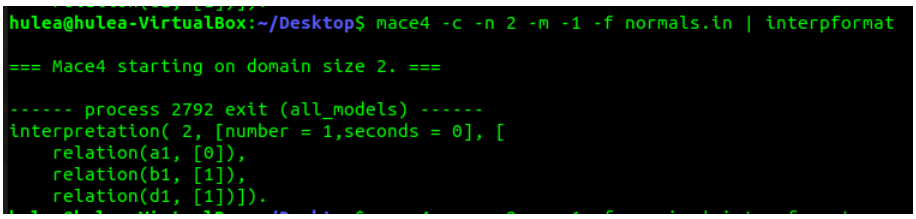

```
hulea@hulea-VirtualBox:~/Desktop$ mace4 -c -n 2 -m -1 -f knightknave.in | interpformat

=== Mace4 starting on domain size 2. ===

------ process 2796 exit (all_models) ------
interpretation( 2, [number = 1,seconds = 0], [
    relation(a1, [0]),
    relation(b1, [0]),
    relation(d1, [1]),
    relation(d2, [1])]).
```

4)Daca A ar fi raspuns da, atunci A ar fi putut fi knight sau normal deoarece nu avem cum sa stim exact daca spune adevarul. Daca A ar fi raspuns nu, atunci A nu ar fi putut fi knight deoarece atunci B ar fi normal si A ar minti, ceea ce nu e posibil. Asadar, singura varianta este ca A sa fi spus nu, deci A este normal.

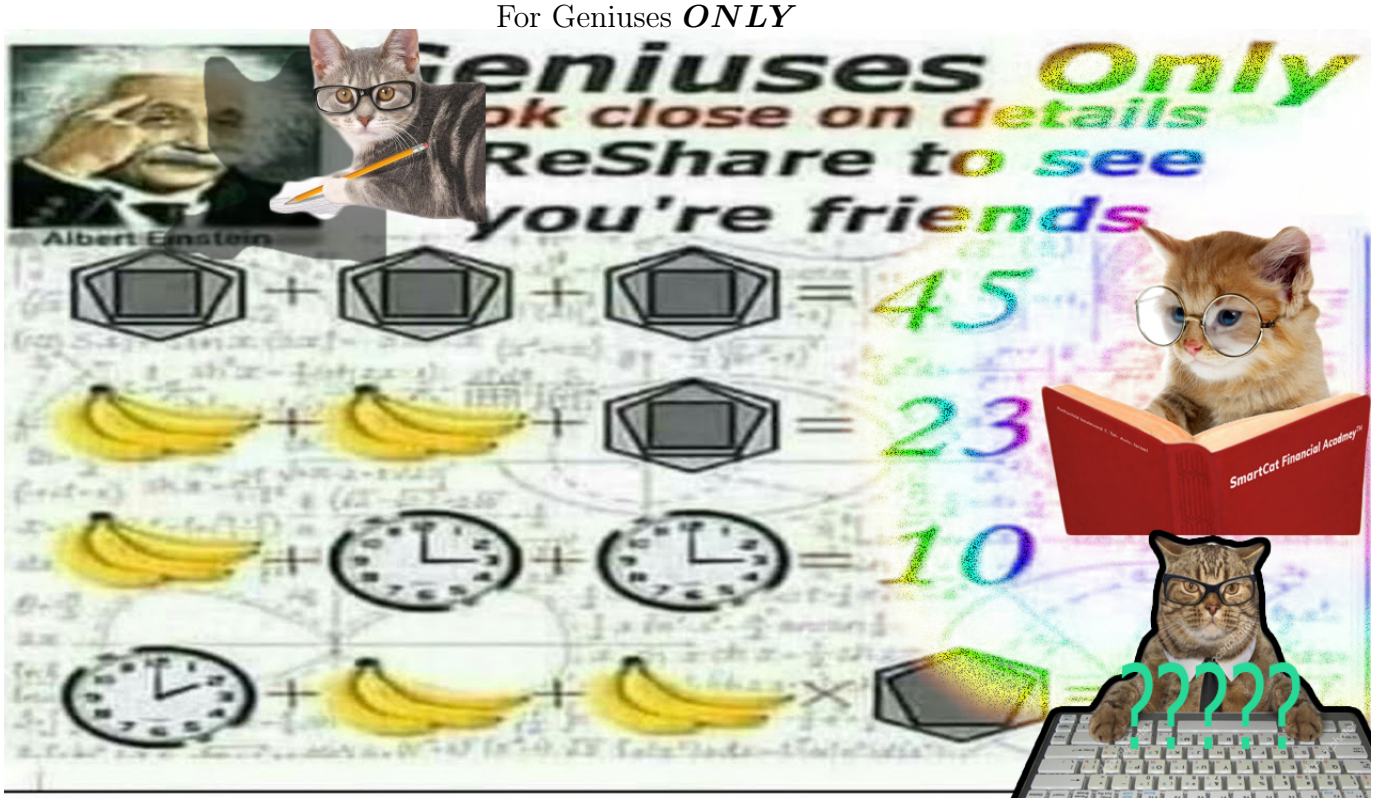



hulea@hulea-VirtualBox:~/Desktop$ mace4 -c -n 2 -m -1 -f normals.in | interpformat
=== Mace4 starting on domain size 2. ===
------ process 2792 exit (all_models) ------
interpretation(2, [number = 1,seconds = 0], [
relation(a1, [0]),
relation(b1, [1]),
relation(d1, [1])]).

2.9 Meme

Ca ultima problema am ales o problema cu un grad de dificultate foarte ridicat, pentru a evidentia inca un mod inedit de a folosi Mace4 si Prove9. Problema este enuntata sub forma unui sistem de ecuatii cu obiecte in loc de necunoscute in imaginea de mai jos.

For Geniuses ***ONLY***



Desigur, si in cazul unei probleme ca aceasta, mace4 ne sara in ajutor, implementarea fiind urmatoarea:

```
set(arithmetic).
formulas(assumptions).
geometric + geometric + geometric = 45.
banana + banana + geometric = 23.
banana + clock + clock = 10.
end_of_list.
```

Rezultatele obtinute:

```
------ process 5954 exit (max_models) ------
interpretation( 16, [number = 1,seconds = 0], [
    function(banana, [4]),
    function(clock, [3]),
    function(geometric, [15])]).
```

Clock + Banana +Banana * Geometric = REZULTAT AICI!

2.10 Conclusion

In concluzie am observat ca programele prover9 si mace4 sunt niste instrumente de folos si complexe deoarece acestea ne-au ajutat sa rezolvam probleme si ne-au evidentiat cat de mult pot usura procesul de rezolvare al unor probleme relativ dificile fata de rezolvarea pe hartie. Acest proiect ne-a ajutat sa intelegem si sa ne antrenam notiuni de logica si creativitatea, fiind in permanenta cautare de rezolvari pentru diverse puzzle-uri logice.
Incercari esuate:

- The Great-grandson Problem by Lewis Carroll
- Sumator Problem
- Minesweeper 20x20
- Sudoku 100x100

Appendix A

Search Code

Listing A.1: DFS

```
def depthFirstSearch(problem):

    primulNod = problem.getStartState()

    if problem.isGoalState(primulNod):
        return []

    stack = util.Stack()
    noduriVizitate = []
    stack.push((primulNod, []))

    while not stack.isEmpty():
        nodAux, directii = stack.pop()
        if nodAux not in noduriVizitate:
            noduriVizitate.append(nodAux)

            if problem.isGoalState(nodAux):
                return directii

            for urmatorul, directie, cost in problem.getSuccessors(nodAux):
                dirNoua = directii + [directie]
                stack.push((urmatorul, dirNoua))
```

Listing A.2: BFS

```
def breadthFirstSearch(problem):
    primulNod = problem.getStartState()
    if problem.isGoalState(primulNod):
        return []

    q = util.Queue()
    noduriVizitate = []

    q.push((primulNod, []))

    while not q.isEmpty():
        nodAux, directii = q.pop()
```

```
        if nodAux not in noduriVizitate:
            noduriVizitate.append(nodAux)

            if problem.isGoalState(nodAux):
                return directii

            for urmatorul, directie, cost in problem.getSuccessors(nodAux):
                dirNoua = directii + [directie]
                q.push((urmatorul, dirNoua))
```

Listing A.3: UCS

```
def uniformCostSearch(problem):
    nodAux = problem.getStartState()
    if problem.isGoalState(nodAux):
        return []

    noduriVizitate = []

    q = util.PriorityQueue()
    q.push((nodAux, [], 0), 0)

    while not q.isEmpty():

        nodAux, directii, prevCost = q.pop()
        if nodAux not in noduriVizitate:
            noduriVizitate.append(nodAux)

            if problem.isGoalState(nodAux):
                return directii

            for urmatorul, directie, cost in problem.getSuccessors(nodAux):
                dirNoua = directii + [directie]
                priority = prevCost + cost
                q.push((urmatorul, dirNoua, priority), priority)
```

Listing A.4: A*

```
def aStarSearch(problem, heuristic=nullHeuristic):
    nodAux = problem.getStartState()
    if problem.isGoalState(nodAux):
        return []

    noduriVizitate = []

    q = util.PriorityQueue()
    q.push((nodAux, [], 0), 0)

    while not q.isEmpty():

        nodAux, directii, prevCost = q.pop()

        if nodAux not in noduriVizitate:
```

```
            noduriVizitate.append(nodAux)

            if problem.isGoalState(nodAux):
                return directii

            for urmatorul, directie, cost in problem.getSuccessors(nodAux):
                dirNoua = directii + [directie]
                costNou = prevCost + cost
                heuristicCost = costNou + heuristic(urmatorul, problem)
                q.push((urmatorul, dirNoua, costNou), heuristicCost)
    util.raiseNotDefined()
```

Listing A.5: Weighted A*

```
def weightedAStarSearch(problem, heuristic=nullHeuristic):
    epsilon = 1.2
    #epsilon 1 a star
    #epsilon 0 ucs
    #epsilon infinit greedy
    nodAux = problem.getStartState()
    if problem.isGoalState(nodAux):
        return []

    noduriVizitate = []

    q = util.PriorityQueue()
    q.push((nodAux, [], 0), 0)

    while not q.isEmpty():

        nodAux, directii, prevCost = q.pop()

        if nodAux not in noduriVizitate:
            noduriVizitate.append(nodAux)

            if problem.isGoalState(nodAux):
                return directii

            for urmatorul, directie, cost in problem.getSuccessors(nodAux):
                dirNoua = directii + [directie]
                costNou = prevCost + cost
                heuristicCost = costNou + epsilon * heuristic(urmatorul, problem)
                q.push((urmatorul, dirNoua, costNou), heuristicCost)
```

Listing A.6: Iterative Deepening DFS

```
def iterativeDeepeningDFS(problem):
    nodAux = problem.getStartState()
    if problem.isGoalState(nodAux):
        return []

    s = util.Stack()
    depth = 30
```

```
    while True:

        noduriVizitate = []
        s.push((problem.getStartState(), [], 0))
        nodAux, directii, prevCost = s.pop()
        noduriVizitate.append(nodAux)

        while not problem.isGoalState(nodAux):
            for urmatorul, directie, cost in problem.getSuccessors(nodAux):
                if urmatorul not in noduriVizitate:
                    if prevCost + cost <= depth:
                        dirNoua = directii + [directie]
                        costNou = prevCost + cost
                        s.push((urmatorul, dirNoua, costNou))
                        noduriVizitate.append(urmatorul)
            if s.isEmpty():
                break
            else:
                nodAux, directii, prevCost = s.pop()

        if problem.isGoalState(nodAux):
            return directii
        else:
            depth = depth + 1
```

Listing A.7: Iterative Deepening A*

```
def iterativeDeepeningAStar(problem):
    from searchAgents import mazeDistance
    nodAux = problem.getStartState()
    if problem.isGoalState(nodAux):
        return []

    s = util.Stack()
    depth = 0

    while True:

        noduriVizitate = []
        s.push((problem.getStartState(), [], 0))
        nodAux, directii, prevCost = s.pop()
        noduriVizitate.append(nodAux)

        while not problem.isGoalState(nodAux):
            for urmatorul, directie, cost in problem.getSuccessors(nodAux):
                if urmatorul not in noduriVizitate:
                    if prevCost + cost <= depth:
                        dirNoua = directii + [directie]
                        costNou = prevCost + cost
                        s.push((urmatorul, dirNoua, costNou))
                        noduriVizitate.append(urmatorul)
            if s.isEmpty():
```

```
                break
            else:
                nodAux, directii, prevCost = s.pop()

        if problem.isGoalState(nodAux):
            return directii
        else:
            depth = depth + mazeDistance(nodAux, problem.goal, problem.state)
```

Listing A.8: Modificari pentru portale in layout.py

```
class Layout:

    def __init__(self, layoutText):
        self.portals = []
        self.width = len(layoutText[0])
        self.height= len(layoutText)
        self.walls = Grid(self.width, self.height, False)
        self.food = Grid(self.width, self.height, False)
        self.capsules = []
        self.agentPositions = []
        self.numGhosts = 0
        self.processLayoutText(layoutText)
        self.layoutText = layoutText
        self.totalFood = len(self.food.asList())
        # self.initializeVisibilityMatrix()

    def processLayoutChar(self, x, y, layoutChar):
        if layoutChar == '%':
            self.walls[x][y] = True
        elif layoutChar == '.':
            self.food[x][y] = True
        elif layoutChar == 'o':
            self.capsules.append((x, y))
        elif layoutChar == 'w':
            self.portals.append((x, y))
        elif layoutChar == 'P':
            self.agentPositions.append( (0, (x, y) ) )
        elif layoutChar in ['G']:
            self.agentPositions.append( (1, (x, y) ) )
            self.numGhosts += 1
        elif layoutChar in ['1', '2', '3', '4']:
            self.agentPositions.append( (int(layoutChar), (x,y)))
            self.numGhosts += 1
```

Listing A.9: Modificari pentru portale in graphicsDisplay.py

```
class PacmanGraphics:
    def drawStaticObjects(self, state):
        layout = self.layout
        self.drawWalls(layout.walls)
        self.food = self.drawFood(layout.food)
```

```
        self.capsules = self.drawCapsules(layout.capsules)
        self.portals = self.drawPortals(layout.portals)
        refresh()

    def update(self, newState):
        agentIndex = newState._agentMoved
        agentState = newState.agentStates[agentIndex]

        self.drawPortals(self.layout.portals)

        if self.agentImages[agentIndex][0].isPacman != agentState.isPacman:
            self.swapImages(agentIndex, agentState)
        prevState, prevImage = self.agentImages[agentIndex]
        if agentState.isPacman:
            self.animatePacman(agentState, prevState, prevImage)
        else:
            self.moveGhost(agentState, agentIndex, prevState, prevImage)
        self.agentImages[agentIndex] = (agentState, prevImage)

        if newState._foodEaten != None:
            self.removeFood(newState._foodEaten, self.food)
        if newState._capsuleEaten != None:
            self.removeCapsule(newState._capsuleEaten, self.capsules)
        self.infoPane.updateScore(newState.score)
        if 'ghostDistances' in dir(newState):
            self.infoPane.updateGhostDistances(newState.ghostDistances)

    def drawPortals(self, portals):
        from random import randrange
        portalImages = {}
        for portal in portals:
            (screen_x, screen_y) = self.to_screen(portal)
            door = circle((screen_x, screen_y),
                          PORTAL_RADIUS * self.gridSize,
                          outlineColor=formatColor((randrange(255))/255.0,
                              (randrange(255))/255.0, (randrange(255))/255.0),
                          fillColor=formatColor((randrange(255))/255.0,
                              (randrange(255))/255.0, (randrange(255))/255.0),
                          width=1)
            door2 = circle((screen_x, screen_y),
                           PORTAL_RADIUS * self.gridSize / 1.2,
                           outlineColor=formatColor((randrange(255))/255.0,
                               (randrange(255))/255.0, (randrange(255))/255.0),
                           fillColor=formatColor((randrange(255))/255.0,
                               (randrange(255))/255.0, (randrange(255))/255.0),
                           width=1)
            door3 = circle((screen_x, screen_y),
                           PORTAL_RADIUS * self.gridSize / 1.7,
                           outlineColor=formatColor((randrange(255)) / 255.0,
                               (randrange(255)) / 255.0,(randrange(255)) / 255.0),
                           fillColor=formatColor((randrange(255))/255.0,
                               (randrange(255))/255.0, (randrange(255))/255.0),
                           width=1)
```

```
            door4 = circle((screen_x, screen_y),
                           PORTAL_RADIUS * self.gridSize / 2.5,
                           outlineColor=formatColor((randrange(255)) / 255.0,
                               (randrange(255)) / 255.0,(randrange(255)) / 255.0),
                           fillColor=formatColor((randrange(255))/255.0,
                               (randrange(255))/255.0, (randrange(255))/255.0),
                           width=1)
            door5 = circle((screen_x, screen_y),
                           PORTAL_RADIUS * self.gridSize / 3.5,
                           outlineColor=formatColor((randrange(255))/255.0,
                               (randrange(255))/255.0, (randrange(255))/255.0),
                           fillColor=formatColor((randrange(255))/255.0,
                               (randrange(255))/255.0, (randrange(255))/255.0),
                           width=1)

            portalImages[portal] = door
            portalImages[portal] = door2
            portalImages[portal] = door3
            portalImages[portal] = door4
            portalImages[portal] = door5

        return portalImages
```

Listing A.10: Modificari pentru portale in game.py

```
class GameStateData:

    def __init__( self, prevState = None ):
        """
        Generates a new data packet by copying information from its predecessor.
        """

        if prevState != None:
            self.food = prevState.food.shallowCopy()
            self.capsules = prevState.capsules[:]
            self.agentStates = self.copyAgentStates( prevState.agentStates )
            self.layout = prevState.layout
            self._eaten = prevState._eaten
            self.score = prevState.score
            self.portals = prevState.portals

        self._foodEaten = None
        self._foodAdded = None
        self._capsuleEaten = None
        self._agentMoved = None
        self._lose = False
        self._win = False
        self.scoreChange = 0
        self._portals = None


    def initialize( self, layout, numGhostAgents ):
```

```
        """
        Creates an initial game state from a layout array (see layout.py).
        """
        self.food = layout.food.copy()
        #self.capsules = []
        self.capsules = layout.capsules[:]
        self.layout = layout
        self.score = 0
        self.scoreChange = 0
        self.portals = layout.portals[:]
        self.agentStates = []
        numGhosts = 0
        for isPacman, pos in layout.agentPositions:
            if not isPacman:
                if numGhosts == numGhostAgents: continue # Max ghosts reached already
                else: numGhosts += 1
            self.agentStates.append( AgentState( Configuration( pos,
                Directions.STOP), isPacman) )
        self._eaten = [False for a in self.agentStates]
```

Listing A.11: Modificari pentru portale in pacman.py

```
class PacmanRules:
    def consume( position, state ):
        x,y = position
        # Eat food
        if state.data.food[x][y]:
            state.data.scoreChange += 10
            state.data.food = state.data.food.copy()
            state.data.food[x][y] = False
            state.data._foodEaten = position
            # TODO: cache numFood?
            numFood = state.getNumFood()
            if numFood == 0 and not state.data._lose:
                state.data.scoreChange += 500
                state.data._win = True
        # Eat capsule
        if position in state.getCapsules():
            state.data.capsules.remove( position )
            state.data._capsuleEaten = position
            # Reset all ghosts' scared timers
            for index in range( 1, len( state.data.agentStates ) ):
                state.data.agentStates[index].scaredTimer = SCARED_TIME

        # Eat portal
        from random import randrange

        if position in state.getPortals():
            portals = state.getPortals()
            #print(len(portals))
            if len(portals) == 2:
                if position == portals[0]:
```

```
                state.data.agentStates[0].configuration.pos = portals[1]
            else:
                state.data.agentStates[0].configuration.pos = portals[0]
        else:
            var = randrange(len(portals))

            while var == position:
                var = randrange((len(portals)))

            state.data.agentStates[0].configuration.pos = portals[var]
            GLOBAL = var

    consume = staticmethod( consume )



class GameState:
    def getPortals(self):
        """
        Returns a list of positions (x,y) of portals
        """
        return self.data.portals
```

Listing A.12: Modificari pentru portale in searchAgents.py

```
class FoodSearchProblem:

    def __init__(self, startingGameState):
        self.start = (startingGameState.getPacmanPosition(),
            startingGameState.getFood())
        self.walls = startingGameState.getWalls()
        self.portals = startingGameState.getPortals()
        self.startingGameState = startingGameState
        self._expanded = 0 # DO NOT CHANGE
        self.heuristicInfo = {} # A dictionary for the heuristic to store information

    def getStartState(self):
        return self.start

    def isGoalState(self, state):
        return state[1].count() == 0

    def getSuccessors(self, state):
        "Returns successor states, the actions they require, and a cost of 1."
        successors = []
        self._expanded += 1 # DO NOT CHANGE
        for direction in [Directions.NORTH, Directions.SOUTH, Directions.EAST,
            Directions.WEST]:
            x, y = state[0]
            dx, dy = Actions.directionToVector(direction)

            nextx, nexty = int(x + dx), int(y + dy)
```

```
            if not self.walls[nextx][nexty]:
                if (nextx, nexty) in self.portals:
                    nextFood = state[1].copy()
                    nextFood[nextx][nexty] = False
                    if (nextx, nexty) == self.portals[0]:
                        successors.append(((self.portals[1], nextFood), direction, 0))
                    else:
                        successors.append(((self.portals[0], nextFood), direction, 0))
                else:
                    nextFood = state[1].copy()
                    nextFood[nextx][nexty] = False
                    successors.append((((nextx, nexty), nextFood), direction, 1))
        return successors



class PositionSearchProblem(search.SearchProblem):

    def __init__(self, gameState, costFn=lambda x: 1, goal=(1, 1), start=None,
        warn=True, visualize=True):

        self.walls = gameState.getWalls()
        self.startState = gameState.getPacmanPosition()
        if start != None: self.startState = start
        self.state = gameState
        self.goal = goal
        self.costFn = costFn
        self.visualize = visualize
        self.portals = gameState.getPortals()
        if warn and (gameState.getNumFood() != 1 or not gameState.hasFood(*goal)):
            print 'Warning: this does not look like a regular search maze'

        # For display purposes
        self._visited, self._visitedlist, self._expanded = {}, [], 0 # DO NOT CHANGE

    def getSuccessors(self, state):

        from pacman import GLOBAL
        from random import randrange
        successors = []
        for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST,
            Directions.WEST]:
            x, y = state
            dx, dy = Actions.directionToVector(action)
            nextx, nexty = int(x + dx), int(y + dy)
            if not self.walls[nextx][nexty]:
                if (nextx, nexty) in self.portals:
                    if (nextx, nexty) == self.portals[0]:
                        successors.append((self.portals[1], action, 0))
                    else:
                        successors.append((self.portals[0], action, 0))
                else:
```

```
                nextState = (nextx, nexty)
                cost = self.costFn(nextState)
                successors.append((nextState, action, cost))

        # Bookkeeping for display purposes
        self._expanded += 1 # DO NOT CHANGE
        if state not in self._visited:
            self._visited[state] = True
            self._visitedlist.append(state)

        return successors
```

Appendix B

Logics Code

Cerinta Among Us

Listing B.1: Among us

```
formulas(assumptions).

color(green).
color(black).
color(purple).
color(blue).
color(yellow).
color(cyan).
color(pink).
color(red).
color(brown).
color(orange).

green_is_the_imposter |
black_is_the_imposter |
purple_is_the_imposter |
blue_is_the_imposter |
cyan_is_the_imposter |
pink_is_the_imposter |
red_is_the_imposter   |
brown_is_the_imposter |
orange_is_the_imposter |
lime_is_the_imposter .


body(x,y) & sus(x,y) & green_location(x,y) -> imposter(x,y).
imposter(x,y) & green_location(x,y) -> green_is_the_imposter.
body(x,y) & sus(x,y) & black_location(x,y) -> imposter(x,y).
imposter(x,y) & black_location(x,y) -> black_is_the_imposter.
body(x,y) & sus(x,y) & purple_location(x,y) -> imposter(x,y).
imposter(x,y) & purple_location(x,y) -> purple_is_the_imposter.
body(x,y) & sus(x,y) & blue_location(x,y) -> imposter(x,y).
imposter(x,y) & blue_location(x,y) -> blue_is_the_imposter.
body(x,y) & sus(x,y) & cyan_location(x,y) -> imposter(x,y).
imposter(x,y) & cyan_location(x,y) -> cyan_is_the_imposter.
body(x,y) & sus(x,y) & pink_location(x,y) -> imposter(x,y).
imposter(x,y) & pink_location(x,y) -> pink_is_the_imposter.
```

```
body(x,y) & sus(x,y) & red_location(x,y) -> imposter(x,y).
imposter(x,y) & red_location(x,y) -> red_is_the_imposter.
body(x,y) & sus(x,y) & brown_location(x,y) -> imposter(x,y).
imposter(x,y) & brown_location(x,y) -> brown_is_the_imposter.
body(x,y) & sus(x,y) & orange_location(x,y) -> imposter(x,y).
imposter(x,y) & orange_location(x,y) -> orange_is_the_imposter.
body(x,y) & sus(x,y) & lime_location(x,y) -> imposter(x,y).
imposter(x,y) & lime_location(x,y) -> lime_is_the_imposter.

green_location(2,2).
black_location(4,5).
purple_location(4,6).
blue_location(4,7).
cyan_location(7,4).
pink_location(8,6).
red_location(0,0).
brown_location(1,0).
orange_location(0,1).
lime_location(2,8).

%EMERGENCY MEETING #1
color(red) <-> (sus(2,2) | sus(3,3)) & body(2,2).
color(black) <-> -(exists x (sus(x,0))).
%EMERGENCY MEETING #2
color(purple) <-> exists x ((sus(x,0)) -> sus(6,x)).
color(cyan) <-> -sus(3,3) .
color(blue) <-> exists x (sus(x,x)).
%EMERGENCY MEETING #4
color(green) <-> sus(8,6).
color(red) <-> -sus(0,0) & -sus(1,0).
%EMERGENCY MEETING #5
color(lime) <-> exists x ((sus(x,5)) -> sus(4,x)).
color(cyan) <-> sus(9,9) | sus(9,8) | body(0,0).
color(orange) <-> body(4,5).
%EMERGENCY MEETING #6
color(purple) <-> body(4,5) & body(4,6) & sus(4,5).
color(brown) <-> sus(4,5) | sus(4,6).
%EMERGENCY MEETING #7
color(red) <-> sus(4,6).
color(cyan) <-> sus(4,6) | -sus(4,6).
color(lime) <-> exists x (body(3,x) -> body(6,3)).
color(pink) <-> body(4,6) & sus(4,6).

end_of_list.

formulas(goals).
black_is_the_imposter & green_is_the_imposter & purple_is_the_imposter.
end_of_list.
```

Cerinta Grid Logic

Listing B.2: Superheroes Grid Logic

```
set(arithmetic).
assign(domain_size,4).
formulas(assumptions).

diff(x,y) -> diff(y,x).

diff( Daredevil, Gorgon).
diff( Daredevil, Loki).
diff( Daredevil, Sandman).
diff( Gorgon, Loki).
diff( Gorgon, Sandman).
diff( Loki, Sandman).

Matthew(x) | Petragon(x) | Laufeyson(x) | William(x).
Superhuman(x) | Hybrid(x) | Giant(x) | Mutant(x).

Matthew(x) & Matthew(y) -> -diff(x,y).
Petragon(x) & Petragon(y) -> -diff(x,y).
Laufeyson(x) & Laufeyson(y) -> -diff(x,y).
William(x) & William(y) -> -diff(x,y).

Superhuman(x) & Superhuman(y) -> -diff(x,y).
Hybrid(x) & Hybrid(y) -> -diff(x,y).
Giant(x) & Giant(y) -> -diff(x,y).
Mutant(x) & Mutant(y) -> -diff(x,y).

year(x) = year(y) -> -diff(x,y).

exists x exists y exists z ( year(x) + 1 = year(y) & Giant(x) & Mutant(y)
              & year(x) + 2 = year(z) & Superhuman(z)).

(year(Sandman) = 1) | Hybrid(Sandman).
Laufeyson(Loki) | William(Loki).
William(Gorgon) | year(Gorgon) = 3.
exists x ( year(Daredevil) = year(x) + 1 & William(x)).
exists x ( (Matthew(Gorgon) & year(x)=2 & Hybrid(x) ) | (Matthew(x) & year(x)=2 &
    Hybrid(Gorgon)) ).
```

Cerinta Schubert

Listing B.3: Utilajul lui Schubert

```
formulas(assumptions).

%1)Ursii, lupii, vulpile, pasarile, gandacii si melcii sunt animale si exista cel
    putin un specimen pentru fiecare.

Urs(x) -> animal(x).
Lup(x) -> animal(x).
Fox(x) -> animal(x).
Pasare(x) -> animal(x).
```

```
Gandac(x) -> animal(x).
Melc(x) -> animal(x).

exists x Urs(x).
exists x Lup(x).
exists x Fox(x).
exists x Pasare(x).
exists x Gandac(x).
exists x Melc(x).

%2)Exista si varza si afine, acestea fiind plante.

exists x Varza(x).
Varza(x) -> planta(x).
exists x Afine(x).
Afine(x) -> planta(x).

%3) Fiecare animal mananca plante sau toate celelalte animale care nu sunt mai mari
    si mananca plante.

all x (animal(x) -> (all y
        (planta(y)->mananca(x,y))
         )
       |
         (all z
          (animal(z) & mai_mare(x,z) &
             (exists u
                (planta(u) & mananca(z,u))) -> mananca(x,z)
          )
         )
     ).

%4) Ursii sunt mai mari decat lupii, care sunt mai mari decat vulpile, care sunt mai
    mari decat pasarile, care sunt mai mari decat gandacii si melcii.
Urs(x) & Lup(y) -> mai_mare(x,y).
Lup(x) & Fox(y) -> mai_mare(x,y).
Fox(x) & Pasare(y) -> mai_mare(x,y).
Pasare(x) & Gandac(y) -> mai_mare(x,y).
Pasare(x) & Melc(y) -> mai_mare(x,y).

%5)Ursii nu mananca lupi,varza si vulpi, lupii nu mananca vulpi sau varza, pasarile
    mananca gandaci dar nu si melci.
Urs(x) & Vulpe(y) -> -mananca(x,y).
Urs(x) & Varza(y) -> -mananca(x,y).
Urs(x) & Lup(y) -> -mananca(x,y).
Lup(x) & Fox(y) -> -mananca(x,y).
Lup(x) & Varza(y) -> -mananca(x,y).
Pasare(x) & Gandac(y) -> mananca(x,y).
Pasare(x) & Melc(y) -> -mananca(x,y).

%6)Ursii mananca afine iar gandacii si melcii mananca unele plante.
Gandac(x) -> (exists y (planta(y) & mananca(x,y))).
Melc(x)      -> (exists y (planta(y) & mananca(x,y))).
```

```
Urs(x)  -> (exists y (planta(y) & mananca(x,y))).

end_of_list.

formulas(goals).

% Exista un animal care mananca toate animalele care mananca varza

exists x exists y ( animal(x) &
              animal(y) &
              mananca(x,y) &
                   (all z (Varza(z) -> mananca(y,z)))).

end_of_list.
```

Cerinta Logic Equations

```
set(arithmetic).
assign(max_models, -1).
assign(domain_size,27).

list(distinct).
   [0,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z].
end_of_list.

formulas(assumptions).

C*G = C.
6*C = C*E.
B + 3 != 2*A.
3*I >= B.
F != A + D.
F = C + H.
3*I <= 9.
C+E >= A.
A+F+H+1 = C + D*I.
E != H + I.
H > B.
J > K.
K=25.
2*L= O.
M=Y+-1.
N+-1=Z + C.
O + 2 = Y.
P= 2 * Z.
Q=L+F.
10*R=10*V + Z.
3*S=3*P + G.
T= U + 1.
U=3*H.
V= T + 1.
2*W = J.
X=B + 2*C + 2.
Y > M & Y<K.
```

```
Z=A+G+2.

end_of_list.
```

Cerinta Who is the spy?

```
formulas(assumptions).
% a1 : a knight
% a2 : a knave
% a3 : a normal
% b1 : b knight
% b2 : b knave
% b3 : b normal
% c1 : c knight
% c2 : c knave
% c3 : c normal

%1 knight 1 knave 1 spy

(a1 & b2 & c3) |
(a1 & b3 & c2) |
(a2 & b1 & c3) |
(a2 & b3 & c1) |
(a3 & b1 & c2) |
(a3 & b2 & c1) .

a1 -> -a2.
a1 -> -a3.
a2 -> -a3.

b1 -> -b2.
b1 -> -b3.
b2 -> -b3.

c1 -> -c2.
c1 -> -c3.
c2 -> -c3.

%a said that c is a knave or c is a spy
da <-> c2 | c3.

%b said that a is a knight or a is a knave or a is the spy
db <-> a1 | a2 | a3.

%c said that b is a knight or b is a knave or b is the spy
dc <-> b1 | b2 | b3.

a1 -> da.
a2 -> -da.
a3 -> da | -da.

b1 -> db.
b2 -> -db.
b3 -> db | -db.
```

```
c1 -> dc.
c2 -> -dc.
c3 -> dc | -dc.

end_of_list.
```

Knight,Knaves and Normals

```
assign(domain_size,4).
formulas(assumptions).
% knight(x) : a knight
% knave(x) : a knave
% normal(x) : a normal
% f1: female from couple 1
% f2: female from couple 2
% b1: male from couple 1
% b2: male from couple 2

% Knight spun totdeauna adevarul si Knave mint
all x ( knight(x) -> true(x)).
all x ( knave(x) -> -true(x)).

Different(x,y) -> Different(y,x).
Different(f1,f2).
Different(f1,b1).
Different(f1,b2).
Different(f2,b1).
Different(f2,b2).
Different(b1,b2).


married(x,y) -> married(y,x).

knave(x) <-> -knight(x) & -normal(x).
knight(x) <-> -knave(x) & -normal(x).
normal(x) <-> -knight(x) & -knave(x).

all x (all y (knave(x) & married(x,y) -> knight(y))).
all x (all y (knight(x) & married(x,y) -> knave(y))).
all x (all y (normal(x) & married(x,y) -> normal(y))).


true(b1) -> knight(b2).
true(f1) -> true(b1) & knight(b2).
true(f2) -> knight(b2).

married(b1,f1).
married(b2,f2).

end_of_list.

formulas(goals).
%normal(x1).
```

```
end_of_list.
```

Cerinta Genius

Rezolvare problema genius only:(3-1) + (4-1) + (4-1) * (15-4) = 38.

Alte metapuzzleuri mai simple

Listing B.4: Ancheta lui Jhon

```
formulas(assumptions).
% j1 = fratele 1 e john
%-j1 = fratele 1 nu e john
% j2 = fratele 2 e john
%-j2 = fratele 2 nu e john

% t1 = fratele 1 zice adv
%-t1 = fratele 1 nu zice adv
% t2 = fratele 2 zice adv
%-t2 = fratele 2 nu zice

%dialog
%d1 = ce spune fratele 1
%d2 = ce spune fratele 2
%%%%%%%%%%%%%%%%%%%%%%%%

%doar un john
(j1 & -j2) | (-j1 & j2).
%cel putin un mincinos
(t1 & -t2) | (-t1 & t2) | (-t1 & -t2).

%dialog
d1 <-> j1.
d2 <-> j2 | -j2.

(-j1 & -t1) -> -d1.
(-j1 & t1) -> -d1.
( j1 & -t1) -> -d1.
( j1 & t1) -> d1.

(-j2 & -t2) -> -d2.
(-j2 & t2) -> -d2.
( j2 & -t2) -> -d2.
( j2 & t2) -> d2.

end_of_list.
```

Alte metapuzzleuri mai simple

Listing B.5: Metapuzzle Transilvania

```
formulas(assumptions).
```

```
%s1/vampire
% s1 : igor nu e nebun
%-s1 : igor e nebun

%sane/insane
% h1 : igor e om
%-h1 : igor e vampir

%dialog
%d1 = ce spune logicianul 1
%d2 = ce spune logicianul 2
%d3 = ce spune logicianul 3
%%%%%%%%%%%%%%%%%%%%%%%

%log1: sane s1?
d1 <-> (h1 & s1) | -(h1 & s1).

%log2 : sane vamp?
d2 <-> (h1 & -s1) | -(h1 & -s1).

%log3 : insane vamp?
d3 <-> (-h1 & -s1) | -(-h1 & -s1).

%aici luam toate cazurile posibile de vampirism si nebunie in urmatorul fel:
%Sane s1 => True
%Insane s1 => False
%Sane Vampire => False
%Insane Vampire => True

( s1 & h1) -> d1.
( s1 & -h1) -> d1.
(-s1 & h1) -> d1.
(-s1 & -h1) -> -d1.


( s1 & h1) -> -d2.
( s1 & -h1) -> d2.
(-s1 & h1) -> -d2.
(-s1 & -h1) -> -d2.

( s1 & h1) -> -d3.
( s1 & -h1) -> d3.
(-s1 & h1) -> d3.
(-s1 & -h1) -> d3.

end_of_list.
```

Alte metapuzzleuri mai simple

Listing B.6: Knight-Knave Metapuzzle

```
 formulas(assumptions).
formulas(assumptions).
% a1 : a knave
```

```
%-a1 : a knight
% b1 : b knave
%-b1 : b knight

%dialog
%d1 = ce spune logicianul 1
%%%%%%%%%%%%%%%%%%%%%%%%

%are you both knights
d1 <-> (a1 & b1) | -(a1 & b1).

%are you two of the same type
d2 <-> ((a1 & b1) | (-a1 & -b1)) | ((-a1 & b1) | (a1 & -b1)).

%aici luam toate cazurile posibile :
%1) a and b knight
%2) a knight, b knave
%3) a knave, b knight
%4) a and b knave

(-a1 & -b1) -> d1.
(-a1 & b1) -> d1.
( a1 & -b1) ->-d1.
( a1 & b1) -> d1.

(-a1 & -b1) -> d2.
(-a1 & b1) ->-d2.
( a1 & -b1) -> d2.
( a1 & b1) ->-d2.

end_of_list.
```

Alte metapuzzleuri mai simple

Listing B.7: Normals Metapuzzle

```
formulas(assumptions).
% a1 : a knight
%-a1 : a normal
% b1 : b knight
%-b1 : b normal

%one knight one knave
(a1 & -b1) | (-a1 & b1).

%is B normal? yes or no
d1 <-> b1 | -d1.

a1 -> d1.
-a1 -> -d1 | d1.

end_of_list.
```

Listing B.8: Great Grandson Problem

```
formulas(assumptions).

Father(x,y) -> Master(y,x).%%%%%%%%%%%%%%%%%%%%%%%%%%
NotSuperior(x,y) & Uncle(x,z) -> -NotOwes(y,z).
Father(x,y) & Enemies(y,z) & Friends(z,w) -> NotOwes(x,w).
Father(x,y) & -NotOwes(y,z) -> Persecutes(z,x).
%NotSuperior(x,y) & Master(x,z) & Father(w,z) -> Senior(y,w).
NotSuperior(x,y) & Master(y,z) & Father(w,z) ->
    Senior(x,w).%%%%%%%%%%%%%%%%%%%%%%%%%%
Father(x,y) & Father(y,z) & -Senior(w,x) -> -Uncle(w,z).
-Master(x,y) & NotSuperior(z,x) & Friends(z,w) & Enemies(w,t) -> -Persecutes(t,y).
Friends(x,y) & -NotSuperior(y,z) & Master(z,w) & Persecutes(t,w) -> Enemies(x,t).
Enemies(x,y) & Persecutes(y,z) & -Master(w,z) & Father(w,t) -> Friends(x,t).

Father(x,y) & Father(y,z) & Father(z,w) <-> Greatgrandson(x,w).

Friends(x,y) -> Friends(y,x).
Enemies(x,y) -> Enemies(y,x).
Friends(x,y) -> -Enemies(x,y).
-Friends(x,y) -> Enemies(x,y).

end_of_list.


formulas(goals).
end_of_list.
```

Listing B.9: Scazator Complet

```
formulas(assumptions).

%daca 2 Terminale sunt conectate, atunci au acelasi semnal

all t1 (all t2 (Terminal(t1) & Terminal(t2) & Connected(t1,t2) -> Signal(t1) =
    Signal(t2) )).

%semnalul la orice Terminal e 1 sau 0
all t (Terminal(t) -> Signal(t)=1 | Signal(t)=0).

%conectarea e comutativa
all t1 ( all t2 ( Connected(t1,t2) <-> Connected(t2,t1))).

%exista 4 tipuri de porti
all g (Gate(g) & k = Type(g) -> (k=AND | k=OR | k=XOR | k=NOT)).


%and e 0 daca si numai daca inputul e 0
all g ( (Gate(g) & (Type(g) = AND)) -> ( (Signal(Out(1,g)) = 0) <-> exists n
    (Signal(In(n,g)) = 0) ) ).
```

```
%or e 1 iff input e 1
all g ( ( Gate(g) & (Type(g) = OR ))-> ( (Signal(Out(1,g)) = 1) <-> exists n
    (Signal(In(n,g)) = 1) ) ).

%xor e 1 iff inputurile sunt diferite
%all g ( ( Gate(g) & (Type(g) = XOR))-> ( (Signal(Out(1,g)) = 1) <-> exists n (
    Signal(In(n,g)) != Signal(In(2,g)) ))).

%not
all g ( Gate(g) & (Type(g) = NOT) -> ( Signal(Out(1,g)) != Signal(In(1,g)) ) ).


%the Gates have 2 inputs and 1 output,except not
all g (Gate(g) & (Type(g) = NOT) -> Arity(g,1,1) ).
all g (Gate(g) & (k = Type(g)) & (k=AND | k=OR | k=XOR) -> Arity(g,2,1)).

%a circuit has Terminals, up to its input and output arity, and nothing beyond its
    arity

all c all i all j ( Circuit(c) & Arity(c,i,j) -> all n ( (n<=i -> Terminal(In(c,n)))
    &
                      (n> i -> In(c,n) = Nothing) &
                      (n<=j -> Terminal(Out(c,n))) &
                      %(n> j -> (c,n) = Nothing)
                      (n> j -> Out(c,n) = Nothing)
                    )).

%Gates, terminals, signals, gate types, and Nothing are all distinct
all g all t ( Gate(g) & Terminal(t) -> (
                g!=t  & g!=1  & g!=0  & g!=OR  & g!=AND  & g!=XOR  & g!=NOT  &
                    g!=Nothing &
                t!=g  & t!=1  & t!=0  & t!=OR  & t!=AND  & t!=XOR  & t!=NOT  &
                    t!=Nothing &
                1!=t  & 1!=g  & 1!=0  & 1!=OR  & 1!=AND  & 1!=XOR  & 1!=NOT  &
                    1!=Nothing &
                0!=t  & 0!=1  & 0!=g  & 0!=OR  & 0!=AND  & 0!=XOR  & 0!=NOT  &
                    0!=Nothing &
                OR!=t & OR!=1 & OR!=0 & OR!=g  & OR!=AND & OR!=XOR & OR!=NOT &
                    OR!=Nothing &
                AND!=t & AND!=1 & AND!=0 & AND!=OR & AND!=g & AND!=XOR & AND!=NOT &
                    AND!=Nothing &
                XOR!=t & XOR!=1 & XOR!=0 & XOR!=OR & XOR!=AND & XOR!=g & XOR!=NOT &
                    XOR!=Nothing &
                NOT!=t & NOT!=1 & NOT!=0 & NOT!=OR & NOT!=AND & NOT!=XOR & NOT!=g &
                    NOT!=Nothing &
                Nothing!=t &
                Nothing!=1 &
                Nothing!=0 &
                Nothing!=OR &
                Nothing!=AND &
                Nothing!=XOR &
                Nothing!=NOT &
                Nothing!=g) ).
```

```
%Gates are circuitss
all g ( Gate(g) -> Circuit(g) ).


Circuit(C1) & Arity(C1,2,1).
%Gate(X1) & Type(X1) = AND.
Gate(X1) & Type(X1) = NOT.
Connected(In(1,C1),In(1,X1)).
%Connected(In(2,C1),In(2,X1)).
Connected(Out(1,X1),Out(1,C1)).

end_of_list.
formulas(assumptions).

  %(Signal(In(1, C1)) = 1) & (Signal(In(2, C1)) = 1) & Signal(Out(1,C1)) = 0.
exists i1 ((Signal(In(1, C1)) = i1) & Signal(Out(1,C1)) = 1).
end_of_list.
```