Artificial intelligence - Project 2
- Dominos and The Resistance -

George Botis
Daria Francioli

10/12/2022

1 Introducere

Scopul acestui proiect este acela de a implementa:

1. un program care rezolvă diferite lvl pentru faimosul joc de Domino.
2. un program care încearcă să rezolve puzzle-ul deducerii rolurilor jucătorilor din jocul „The Resistance”.

1.1 Domino

Domino este un joc celebru cu piese de domino. Doi sau mai mulți jucători trebuie să pună alternativ piesele potrivite. Câștigătorul este persoana care scapă mai întâi de toate piesele sale.

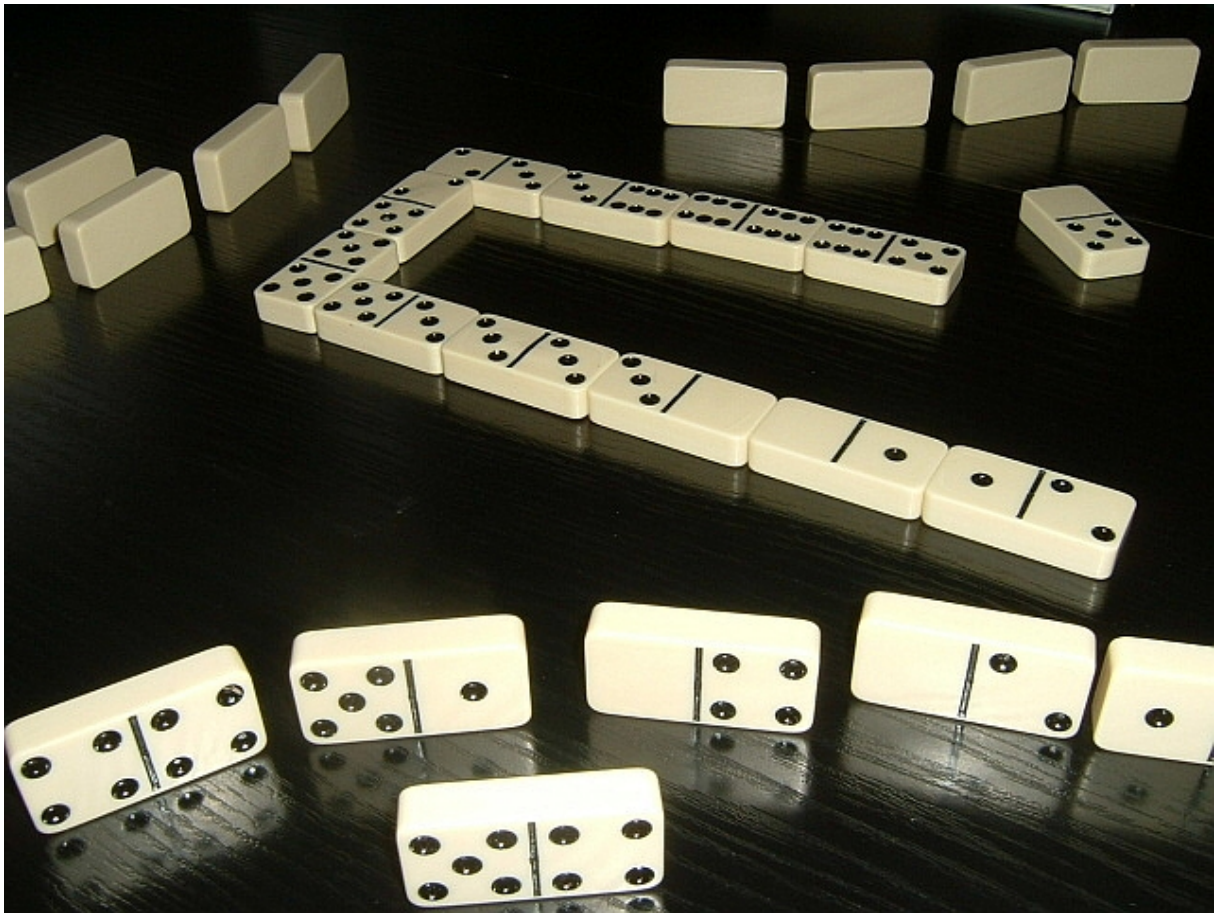

Figure 1: Domino

Avem nevoie de opt secvențe de șapte pătrate pentru piese. În total sunt 56 de pătrate. Ele reprezintă numerele de la 0 la 6. Numerele sunt reprezentate de puncte ca pe zarurile de joc. Un pătrat gol ia locul numărului 0.

1.2 The Resistance

The Resistance este un joc de deducție socială care implică jucătorii care încearcă să-și dea seama cine dintre ei face parte din „Rezistență” și cine face parte din echipa „Guvernul”. Jocul este plasat într-o lume fictivă în care Rezistența încearcă să răstoarne un guvern tiranic, iar echipa de spionaj lucrează pentru a le sabota eforturile.

Fiecărui jucător i se atribuie în secret un rol la începutul jocului, iar scopul jocului este ca echipa Rezistenței să finalizeze cu succes o serie de misiuni fără a fi detectată de echipa de spionaj. Echipa de spionaj, pe de altă parte, vrea să împiedice Rezistența să reușească votând împotriva propunerilor de misiune și încercând să-și dea seama cine sunt ceilalți spioni.

Figure 2: The Resistance Board Game

Mai avem si jucători de tip Trădători, în echipa de spioni care lucrează în secret împotriva celorlalți din guvern. Acest jucător are propriile obiective secrete pe care trebuie să încerce să le ducă la bun sfârșit, încercând, de asemenea, să se îmbine cu ceilalți spioni.

Trădătorul este o răsturnare a jocului tradițional Resistance care adaugă un strat suplimentar de înșelăciune, intrigă și face și jocul mai complex.

2 Definirea problemei

În această secțiune vom prezenta problma fiecărui puzzle.

2.1 Domino - Seven Squares

1. Prima Problemă

   Așezarea a șapte cadre pătrate cu toate cele 28 de piese de domino. Dominourile cu același număr de puncte se întâlnesc.

2. A doua Problemă

   Așezarea a șapte cadre pătrate cu toate cele 28 de piese de domino, astfel încât sumele numerelor de puncte să fie aceleași pe toate cele patru laturi.
   Suma este $4 + 4 + 2 = 2 + 2 + 6 = 6 + 1 + 3 = 3 + 3 + 4 = 10$.

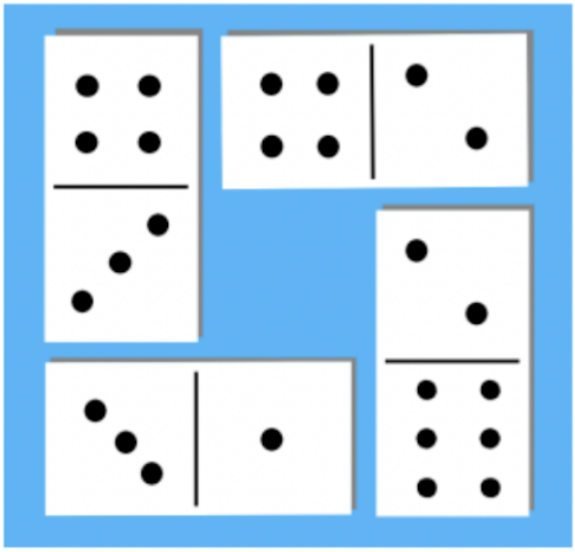

Figure 3: Seven Squares

După cum se observă și în poză, fiecare 2 capete vecine 3-3 , 4-4 și 2-2, au același număr. Dar, 6!=1. Se poate găsi o posibilitate pentru a îndeplini toate condițiile? Vom afla la implementare.

2.2 The Resistance - Spy View

1. Prima Problemă

   Din punctul de vedere al echipei de spioni, cea mai mare problemă din jocul este încercarea de a afla cine sunt ceilalți spioni. Pentru a avea succes, echipa de spionaj trebuie să lucreze împreună și să-și coordoneze eforturile fără a se lăsa în fața echipei Rezistența.

2. A doua Problemă

   O altă problemă pentru echipa de spionaj este să încerce să saboteze misiunile echipei de Rezistență fără a fi prins. Echipa de spionaj își poate folosi voturile pentru a încerca să blocheze propunerile de misiune, dar dacă sunt prea evidente, echipa Rezistenței poate deveni suspicioasă și poate începe să-și dea seama cine sunt spionii. Aceasta înseamnă că echipa de spionaj trebuie să fie atentă și strategică în modul în care își folosesc voturile și să încerce să saboteze misiunile.

În general, echipa de spionaj se confruntă cu o serie de provocări în jocul Rezistenței, inclusiv încercarea de a-și da seama cine sunt colegii lor spioni, lucrul împreună fără a se dezvălui și sabotarea eforturilor echipei Rezistență fără a fi prins. Aceste provocări pot face jocul captivant și plin de suspans și necesită ca echipa de spionaj să-și folosească inteligența și gândirea strategică pentru a reuși.

3 Implementare

Seven Squares problema 1:

Cod:

```
assign(domain_size, 7).
assign(max_models, -1).
set(arithmetic).

formulas(assumption).
c1=c8 & c2=c3 & c4=c5 & c6=c7.

end_of_list.

formulas(goal).
-((c1 = c3 | c1 = c4) & (c2 = c3 | c2 = c4)).
-((c1 = c5 | c1 = c6) & (c2 = c5 | c2 = c6)).
-((c1 = c7 | c1 = c8) & (c2 = c7 | c2 = c8)).
-((c3 = c5 | c3 = c6) & (c4 = c5 | c4 = c6)).
-((c3 = c7 | c3 = c8) & (c4 = c7 | c4 = c8)).
-((c5 = c7 | c5 = c8) & (c6 = c7 | c6 = c8)).
end_of_list.
```

Seven Squares problema 2:

Cod:

```
assign(domain_size, 7).
assign(max_models, -1).
set(arithmetic).

formulas(assumption).
c1+c2+c3=c3+c4+c5 & c3+c4+c5=c5+c6+c7 & c5+c6+c7=c7+c8+c1.

end_of_list.

formulas(goal).
-(c1 = c3 | c1 = c4 | c2 = c3 | c2 = c4).
-(c1 = c5 | c1 = c6 | c2 = c5 | c2 = c6).
-(c1 = c7 | c1 = c8 | c2 = c7 | c2 = c8).
-(c3 = c5 | c3 = c6 | c4 = c5 | c4 = c6).
-(c3 = c7 | c3 = c8 | c4 = c7 | c4 = c8).
-(c5 = c7 | c5 = c8 | c6 = c7 | c6 = c8).
end_of_list.
```

The Resistance:

Cod:

```
% constants: a, b, c, d, e, f (the players)

% predicate Resistance(x): x is a resistance
% predicate Spy(x): x is a spy
% predicate Traitor(x): x is a traitor
% predicate Government(x): x is a government

% predicate Knows(x,y): x knows information y
% predicate Dead(x): x is dead
% predicate Attacked(x,y): x attacked y
% predicate Criminal(x): x is a criminal

set(print_gen).

formulas(assumptions).

spy(a).
resistance(b).
resistance(c).
traitor(d).
goverment(e).
goverment(f).

IsSpy(a, b, c, d) -> Knows(a, a) & Knows(a, b) & Knows(a, c) & Knows(a, e) & Knows(a, f).

Traitor(a, d, e, f) -> Attacked(d,e) & Attacked(d, a) & Attacked(d, f).

-Knows(b, a).
-Knows(b, d).
-Knows(c, a).
-Knows(c, d).
-Knows(d, a).
-Knows(d, b).
-Knows(d, c).

-Attacked(a, d).
Attacked(x, y) -> Dead(y).

Attacked(d, e) | Attacked(d, f) -> Knows(a, d).
Knows(a, d) -> Criminal(d).
Criminal(d) -> Attacked(a, d).

Dead(e) & Dead(f) -> ResistanceVictory(b, c, d).
Dead(d) -> GovermentVictory(a, e, f).

end_of_list.

formulas(goals).
ResistanceVictory(b, c, d).
end_of_list.
```

4 Soluțiile problemelor

Seven Squares problema 1:

```
============================== MODEL ================================

interpretation( 7, [number=1, seconds=0], [

        function(c1, [ 0 ]),

        function(c2, [ 1 ]),

        function(c3, [ 1 ]),

        function(c4, [ 2 ]),

        function(c5, [ 2 ]),

        function(c6, [ 0 ]),

        function(c7, [ 0 ]),

        function(c8, [ 0 ])
]).

============================== end of model ==========================
============================== MODEL ================================

interpretation( 7, [number=35, seconds=0], [

        function(c1, [ 0 ]),

        function(c2, [ 2 ]),

        function(c3, [ 2 ]),

        function(c4, [ 3 ]),

        function(c5, [ 3 ]),

        function(c6, [ 6 ]),

        function(c7, [ 6 ]),

        function(c8, [ 0 ])
]).

============================== end of model ==========================
```

```
============================== MODEL ================================

interpretation( 7, [number=485, seconds=0], [

        function(c1, [ 3 ]),

        function(c2, [ 1 ]),

        function(c3, [ 1 ]),

        function(c4, [ 2 ]),

        function(c5, [ 2 ]),

        function(c6, [ 6 ]),

        function(c7, [ 6 ]),

        function(c8, [ 3 ])
]).

============================== end of model ==========================
============================== MODEL ================================

interpretation( 7, [number=1016, seconds=0], [

        function(c1, [ 6 ]),

        function(c2, [ 4 ]),

        function(c3, [ 4 ]),

        function(c4, [ 3 ]),

        function(c5, [ 3 ]),

        function(c6, [ 0 ]),

        function(c7, [ 0 ]),

        function(c8, [ 6 ])
]).

============================== end of model ==========================
```

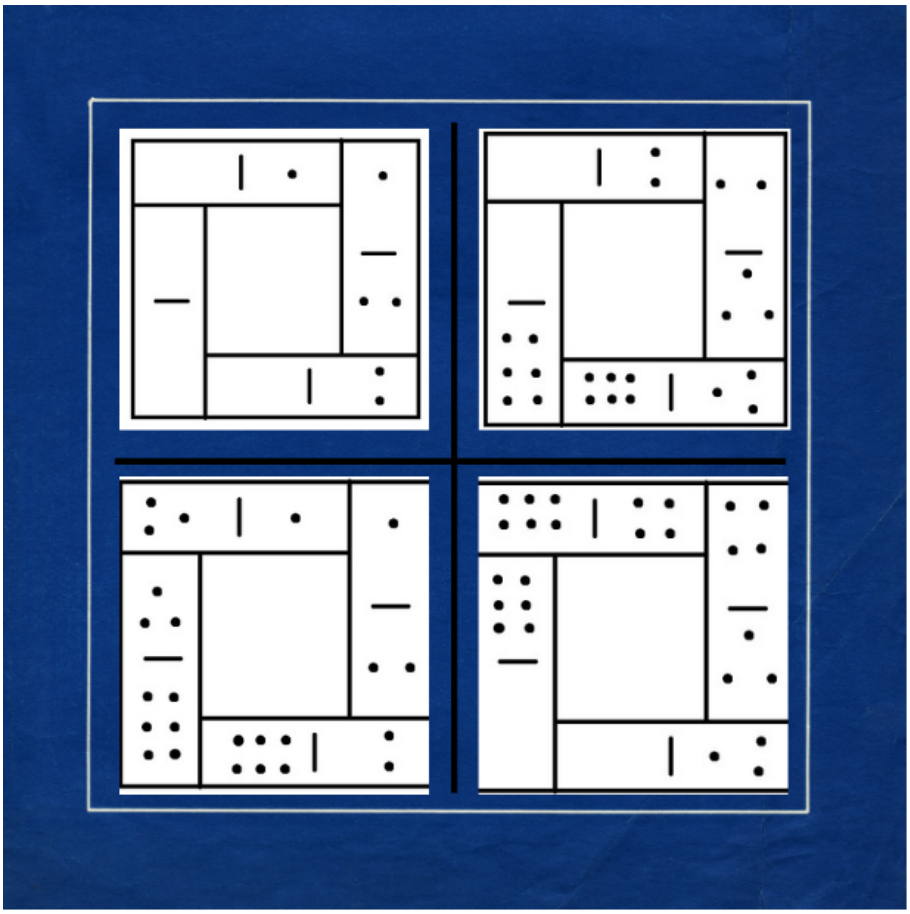

Figure 4: Seven Squares 1

Am creat o vizualizare clară a celor 4 exemple alese random pentru prima problemă a dominourilor. Se poate observa cu ușurință că se respectă cerințele problemei și că fiecare 2 vecini au aceleași numere.

Seven Squares problema 2:

```
============================== MODEL ================================

interpretation( 7, [number=7, seconds=0], [

        function(c1, [ 0 ]),

        function(c2, [ 1 ]),

        function(c3, [ 4 ]),

        function(c4, [ 0 ]),

        function(c5, [ 1 ]),

        function(c6, [ 2 ]),

        function(c7, [ 2 ]),

        function(c8, [ 3 ])
]).

============================== end of model ==========================

============================== MODEL ================================

interpretation( 7, [number=8, seconds=0], [

        function(c1, [ 0 ]),

        function(c2, [ 1 ]),

        function(c3, [ 4 ]),

        function(c4, [ 0 ]),

        function(c5, [ 1 ]),

        function(c6, [ 3 ]),

        function(c7, [ 1 ]),

        function(c8, [ 4 ])
]).

============================== end of model ==========================
```

```
============================== MODEL ================================

interpretation( 7, [number=9, seconds=0], [

        function(c1, [ 0 ]),

        function(c2, [ 1 ]),

        function(c3, [ 4 ]),

        function(c4, [ 0 ]),

        function(c5, [ 1 ]),

        function(c6, [ 4 ]),

        function(c7, [ 0 ]),

        function(c8, [ 5 ])
]).

============================== end of model ==========================

============================== MODEL ================================

interpretation( 7, [number=10, seconds=0], [

        function(c1, [ 0 ]),

        function(c2, [ 1 ]),

        function(c3, [ 4 ]),

        function(c4, [ 1 ]),

        function(c5, [ 0 ]),

        function(c6, [ 2 ]),

        function(c7, [ 3 ]),

        function(c8, [ 2 ])
]).

============================== end of model ==========================









============================== MODEL ================================
```

```
interpretation( 7, [number=11, seconds=0], [

        function(c1, [ 0 ]),

        function(c2, [ 1 ]),

        function(c3, [ 4 ]),

        function(c4, [ 1 ]),

        function(c5, [ 0 ]),

        function(c6, [ 3 ]),

        function(c7, [ 2 ]),

        function(c8, [ 3 ])
]).

============================== end of model ==========================

============================== MODEL =================================

interpretation( 7, [number=12, seconds=0], [

        function(c1, [ 0 ]),

        function(c2, [ 1 ]),

        function(c3, [ 5 ]),

        function(c4, [ 0 ]),

        function(c5, [ 1 ]),

        function(c6, [ 1 ]),

        function(c7, [ 4 ]),

        function(c8, [ 2 ])
]).

============================== end of model ==========================
```

```
160 ============================== MODEL ================================
161
162 interpretation( 7, [number=13, seconds=0], [
163
164         function(c1, [ 0 ]),
165
166         function(c2, [ 1 ]),
167
168         function(c3, [ 5 ]),
169
170         function(c4, [ 0 ]),
171
172         function(c5, [ 1 ]),
173
174         function(c6, [ 2 ]),
175
176         function(c7, [ 3 ]),
177
178         function(c8, [ 3 ])
179 ]).
180
181 ============================== end of model ==========================
```

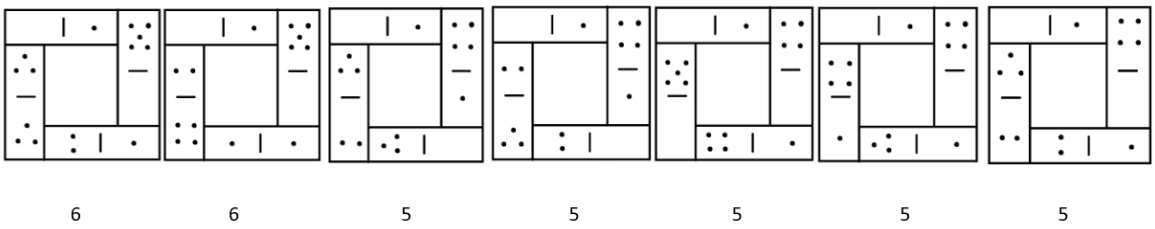

Figure 5: Seven Squares 2

The Resistance:

```
============================== INPUT =====================================

% Reading from file ress.in

set(print_gen).

formulas(assumptions).
spy(a).
resistance(b).
resistance(c).
traitor(d).
goverment(e).
goverment(f).
IsSpy(a,b,c,d) -> Knows(a,a) & Knows(a,b) & Knows(a,c) & Knows(a,e) & Knows(a,f).
Traitor(a,d,e,f) -> Attacked(d,e) & Attacked(d,a) & Attacked(d,f).
-Knows(b,a).
-Knows(b,d).
-Knows(c,a).
-Knows(c,d).
-Knows(d,a).
-Knows(d,b).
-Knows(d,c).
-Attacked(a,d).
Attacked(x,y) -> Dead(y).
Attacked(d,e) | Attacked(d,f) -> Knows(a,d).
Knows(a,d) -> Criminal(d).
Criminal(d) -> Attacked(a,d).
Dead(e) & Dead(f) -> ResistanceVictory(b,c,d).
Dead(d) -> GovermentVictory(a,e,f).
end_of_list.

formulas(goals).
ResistanceVictory(b,c,d).
end_of_list.

============================== end of input ==========================

============================== PROCESS NON-CLAUSAL FORMULAS ==========

% Formulas that are not ordinary clauses:
1 IsSpy(a,b,c,d) -> Knows(a,a) & Knows(a,b) & Knows(a,c) & Knows(a,e) & Knows(a,f) # label(non_clause).
2 Traitor(a,d,e,f) -> Attacked(d,e) & Attacked(d,a) & Attacked(d,f) # label(non_clause).  [assumption].
3 Attacked(x,y) -> Dead(y) # label(non_clause).  [assumption].
4 Attacked(d,e) | Attacked(d,f) -> Knows(a,d) # label(non_clause).  [assumption].
5 Knows(a,d) -> Criminal(d) # label(non_clause).  [assumption].
6 Criminal(d) -> Attacked(a,d) # label(non_clause).  [assumption].
7 Dead(e) & Dead(f) -> ResistanceVictory(b,c,d) # label(non_clause).  [assumption].
8 Dead(d) -> GovermentVictory(a,e,f) # label(non_clause).  [assumption].
9 ResistanceVictory(b,c,d) # label(non_clause) # label(goal).  [goal].

============================== end of process non-clausal formulas ===
```

```
============================== PROCESS INITIAL CLAUSES ===============

% Clauses before input processing:

formulas(usable).
end_of_list.

formulas(sos).
spy(a).  [assumption].
resistance(b).  [assumption].
resistance(c).  [assumption].
traitor(d).  [assumption].
goverment(e).  [assumption].
goverment(f).  [assumption].
-IsSpy(a,b,c,d) | Knows(a,a).  [clausify(1)].
-IsSpy(a,b,c,d) | Knows(a,b).  [clausify(1)].
-IsSpy(a,b,c,d) | Knows(a,c).  [clausify(1)].
-IsSpy(a,b,c,d) | Knows(a,e).  [clausify(1)].
-IsSpy(a,b,c,d) | Knows(a,f).  [clausify(1)].
-Traitor(a,d,e,f) | Attacked(d,e).  [clausify(2)].
-Traitor(a,d,e,f) | Attacked(d,a).  [clausify(2)].
-Traitor(a,d,e,f) | Attacked(d,f).  [clausify(2)].
-Knows(b,a).  [assumption].
-Knows(b,d).  [assumption].
-Knows(c,a).  [assumption].
-Knows(c,d).  [assumption].
-Knows(d,a).  [assumption].
-Knows(d,b).  [assumption].
-Knows(d,c).  [assumption].
-Attacked(a,d).  [assumption].
-Attacked(x,y) | Dead(y).  [clausify(3)].
-Attacked(d,e) | Knows(a,d).  [clausify(4)].
-Attacked(d,f) | Knows(a,d).  [clausify(4)].
-Knows(a,d) | Criminal(d).  [clausify(5)].
-Criminal(d) | Attacked(a,d).  [clausify(6)].
-Dead(e) | -Dead(f) | ResistanceVictory(b,c,d).  [clausify(7)].
-Dead(d) | GovermentVictory(a,e,f).  [clausify(8)].
-ResistanceVictory(b,c,d).  [deny(9)].
end_of_list.

formulas(demodulators).
end_of_list.

============================== PREDICATE ELIMINATION =================

Eliminating spy/1

Eliminating resistance/1

Eliminating traitor/1

Eliminating goverment/1
```

```
Eliminating IsSpy/4

Eliminating Traitor/4

Eliminating Knows/2
10 -Attacked(d,e) | Knows(a,d).  [clausify(4)].
11 -Knows(b,a).  [assumption].
12 -Knows(b,d).  [assumption].
13 -Knows(c,a).  [assumption].
14 -Knows(c,d).  [assumption].
15 -Knows(d,a).  [assumption].
16 -Knows(d,b).  [assumption].
17 -Knows(d,c).  [assumption].
18 -Attacked(d,f) | Knows(a,d).  [clausify(4)].
19 -Knows(a,d) | Criminal(d).  [clausify(5)].
Derived: Criminal(d) | -Attacked(d,e).  [resolve(19,a,10,b)].
Derived: Criminal(d) | -Attacked(d,f).  [resolve(19,a,18,b)].

Eliminating Attacked/2
20 -Criminal(d) | Attacked(a,d).  [clausify(6)].
21 -Attacked(a,d).  [assumption].
22 -Attacked(x,y) | Dead(y).  [clausify(3)].
Derived: -Criminal(d).  [resolve(20,b,21,a)].
23 Criminal(d) | -Attacked(d,e).  [resolve(19,a,10,b)].
24 Criminal(d) | -Attacked(d,f).  [resolve(19,a,18,b)].

Eliminating ResistanceVictory/3
25 -ResistanceVictory(b,c,d).  [deny(9)].
26 -Dead(e) | -Dead(f) | ResistanceVictory(b,c,d).  [clausify(7)].
Derived: -Dead(e) | -Dead(f).  [resolve(25,a,26,c)].

Eliminating GovermentVictory/3

Eliminating Criminal/1

============================== end predicate elimination =============

Auto_denials:  (no changes).

Term ordering decisions:
Predicate symbol precedence:  predicate_order([ Dead ]).
Function symbol precedence:  function_order([ e, f ]).
After inverse_order:  (no changes).
Unfolding symbols: (none).
```

5 Conluzii

Prover9 și Mace4 pot fi folosite pentru a rezolva diferite tipuri de puzzle-uri și jocuri sociale. În general, acești demonstratori de teoreme funcționează luând un set de ipoteze logice și o formulă de obiectiv ca intrare și apoi încercând să găsească o soluție care să satisfacă ipotezele și obiectivul.

Pentru a utiliza Prover9 și Mace4 pentru puzzle-uri și jocuri sociale, primul pas este să exprimăm regulile și constrângerile puzzle-ului sau jocului ca formule logice.

Odată ce am exprimat regulile și constrângerile puzzle-ului sau jocului ca formule logice, putem folosi Prover9 sau Mace4 pentru a găsi o soluție care să satisfacă aceste formule.