Τεχνητή Νοημοσύνη 2

Εργαστήριο ASP

Lab 1

```
%---------- NODE COLORING ----------%

% FACTS
node(1..6).
edge(1, 2), edge(2, 4), edge(3, 1).
edge(4, 1), edge(5, 3), edge(6, 2).
edge(1, 3), edge(2, 5), edge(3, 4).
edge(4, 2), edge(5, 4), edge(6, 3).
edge(1, 4), edge(2, 6), edge(3, 5).
edge(5, 6), edge(6, 5).
col(r). col(g). col(b).

% RULE
1 {color(X, Z) : col(Z)} 1 :- node(X).

% INTEGRITY CONSTRAINT
:- color(X, Z), color(Y, Z), edge(X, Y).

% OUTPUT
#show color/2.
```

```
% - - - - - - - - - - -  EINSTEIN'S RIDDLE  - - - - - - - - - - %

%*
Conditions:
There are 5 houses painted five different colors.
In each house lives a person with a different nationality.
These five owners drink a certain type of beverage, smoke a certain brand of
cigar, and keep a certain pet.
No owners have the same pet, smoke the same brand of cigar, or drink the same
beverage.

Details:
1)The Brit lives in the red house
2)The Swede keeps dogs as pets
```

```
3)The Dane drinks tea
4)The green house is on the left of the white house
5)The person who smokes Pall Malls rears birds
6)The owner of the yellow house smokes Dunhill
7)The green house’s owner drinks coffee
8)The man living in the center house drinks milk
9)The Norwegian lives in the first (leftmost) house
10)The man who smokes Blends lives next to the one who keeps cats
11)The man who keeps horses lives next to the man who smokes Dunhill
12)The owner who smokes BlueMaster drinks beer
13)The German smokes Princes
14)The Norwegian lives next to the blue house
15)The man who smokes Blends has a neighbor who drinks water

Question:
Now to solve, tell me who owns the fish?
*%

% FACTS
% There are 5 houses.
house(1..5).
%* The houses are painted with one of the five colors:
red, green, blue, white, yellow *%
col(r; g; b; w; y).
%*In each house lives a person with one of the following nationalities:
Brit, Swede, German, Norwegian, Dane*%
man(b; s; g; n; d).
%* These five owners drink one of the following beverages:
coffee, water, milk, beer, tea*%
drink(c; w; m; b; t).
%* These five owners smoke one of the following brands:
Princes, BlueMaster, Pall Malls, Dunhill, Blends*%
cigarretes(p; bm; pm; d; b).
%* These five owners has one of the following pets:
cats, dogs, birds, fish, horses*%
pet(c; d; b; f; h).

% RULES
%* Formulate the relations:
lives, hasColor, smokes, drinks and hasPet *%
1 {lives(X, Y) : house(Y)} 1 :-  man(X).
1 {hasColor(X, Y) : col(Y)} 1 :-  house(X).
1 {smokes(X, Y) : cigarretes(Y)} 1 :-  man(X).
1 {drinks(X, Y) : drink(Y)} 1 :-  man(X).
1 {hasPet(X, Y) : pet(Y)} 1 :-  man(X).
```

```
% INTEGRITY CONSTRAINTS
%* Formulate that no two owners live on the same house, smoke the same brand of
cigar, drink the same beverage
or have the same pet and also that each house has a unique color.*%
:-  lives(X, Y), lives(Z, Y), X != Z.
:-  smokes(X, Y), smokes(Z, Y), X != Z.
:-  drinks(X, Y), drinks(Z, Y), X != Z.
:-  hasPet(X, Y), hasPet(Z, Y), X != Z.
:-  hasColor(X, Y), hasColor(Z, Y), X != Z.
%* The Brit lives in the red house *%
:-  lives(b, Y), hasColor(Y, Z), Z != r.
%* The Swede keeps dogs as pets *%
:-  hasPet(s, Y), Y != d.
%* The Dane drinks tea *%
:-  drinks(d, Y), Y != t.
%* The green house is on the left of the white house *%
:-  hasColor(X, g), hasColor(Y, w), X != Y - 1.
%* The green house’s owner drinks coffee *%
:-  lives(X, Y), hasColor(Y, g), drinks(X, Z), Z != c.
%* The person who smokes Pall Malls rears birds *%
:-  smokes(X, pm), hasPet(X, Y), Y != b.
%* The owner of the yellow house smokes Dunhill *%
:-  lives(X, Y), hasColor(Y, y), smokes(X, Z), Z != d.
%* The man living in the center house drinks milk *%
:-  drinks(X, m), lives(X, Y), Y != 3.
%* The Norwegian lives in the first (leftmost) house *%
:-  lives(n, Y), Y != 1.
%* The man who smokes Blends lives next to the one who keeps cats *%
:-  smokes(X, b), lives(X, Y), hasPet(Z, c), lives(Z, W), |Y - W| != 1.
%* The man who keeps horses lives next to the man who smokes Dunhill *%
:-  smokes(X, d), lives(X, Y), hasPet(Z, h), lives(Z, W), |Y - W| != 1.
%* The owner who smokes BlueMaster drinks beer *%
:-  smokes(X, bm), drinks(X, Y), Y != b.
%* The German smokes Princes *%
:-  smokes(g, Y), Y != p.
%* The Norwegian lives next to the blue house *%
:-  lives(n, Y), hasColor(Z, b), |Y - Z| != 1.
%* The man who smokes Blends has a neighbor who drinks water *%
:-  smokes(X, b), lives(X, Y), drinks(Z, w), lives(Z, W), |Y - W| != 1.

% OUTPUT
#show lives/2.
#show hasColor/2.
#show smokes/2.
```

```
#show drinks/2.
#show hasPet/2.

%*
Results:

  Brit     Swede     German     Norwegian    Dane
lives(b, 3)  |lives(s, 5)  | lives(g, 4)  | lives(n, 1)  | lives(d, 2)
hasColor(3, r) |hasColor(5, w) | hasColor(4, g) | hasColor(1, y) | hasColor(2, b)
smokes(b, pm) |smokes(s, bm) | smokes(g, p)  | smokes(n, d)  | smokes(d, b)
drinks(b, m)  |drinks(s, b)  | drinks(g, c)  | drinks(n, w)  | drinks(d, t)
hasPet(b, b)  |hasPet(s, d)  | hasPet(g, f)  | hasPet(n, c)  | hasPet(d, h)

Answer: The Norwegian owns the fish
*%
```

Lab 2

```
%---------- 4 TOWERS AND 4 BISHOPS ----------%

% RULES
1 {tower(I, X, Y) : X = 1..8, Y = 1..8} 1 :- I = 1..4.
1 {bishop(I, X, Y) : X = 1..8, Y = 1..8} 1 :- I = 1..4.
occupied(Y) :- tower(_, _, Y), Y = 1..8.
occupied(Y) :- bishop(_, _, Y), Y = 1..8.

% INTEGRITY CONSTRAINTS
:- tower(I, X, Y1), tower(J, X, Y2), I != J.
:- tower(I, X1, Y), tower(J, X2, Y), I != J.
:- bishop(I, X1, Y1), bishop(J, X2, Y2), X1 + X2 == Y1 + Y2, I!=J.
:- bishop(I, X1, Y1), bishop(J, X2, Y2), X1 + Y1 == X2 + Y2, I!=J.
:- tower(I, X, Y1), bishop(J, X, Y2).
:- tower(I, X1, Y), bishop(J, X2, Y).
:- tower(I, X1, Y1), bishop(J, X2, Y2), X1 + Y1 == X2 + Y2.
:- tower(I, X1, Y1), bishop(J, X2, Y2), X1 + Y2 == X2 + Y1.
:- not occupied(Y), Y = 1..8.

% OUTPUT
#show bishop/3.
#show tower/3.
```

Lab 3

```
%---------- SUDOKU SOLVER ----------%

% BOX
#const n = 3.

% FACTS
% Puzzle description (input) -> fixed(x_pos,  y_pos,  value)
fixed(1, 1, 5). fixed(1, 2, 3). fixed(1, 5, 7).
fixed(2, 1, 6). fixed(2, 4, 1). fixed(2, 5, 9). fixed(2, 6, 5).
fixed(3, 2, 9). fixed(3, 3, 8). fixed(3, 8, 6).
fixed(4, 1, 8). fixed(4, 5, 6). fixed(4, 9, 3).
fixed(5, 1, 4). fixed(5, 4, 8). fixed(5, 6, 3). fixed(5, 9, 1).
fixed(6, 1, 7). fixed(6, 5, 2). fixed(6, 9, 6).
fixed(7, 2, 6). fixed(7, 7, 2). fixed(7, 8, 8).
fixed(8, 4, 4). fixed(8, 5, 1). fixed(8, 6, 9). fixed(8, 9, 5).
fixed(9, 5, 8). fixed(9, 8, 7). fixed(9, 9, 9).

% RULES
1 {value(X, Y, I) : I = 1..n * n}  1 :- X = 1..n * n , Y = 1..n * n .
samebox(X1, Y1, X2, Y2) :- X1 = 1..n * n, X2 = 1..n * n, Y1 = 1..n * n, Y2 = 1..n
* n, (X1 - 1) / n  =  (X2 - 1) / n, (Y1 - 1) / n  =  (Y2 - 1) / n.

% INTEGRITY CONSTRAINTS
:- fixed(X, Y, I1), value(X, Y, I2), I1 != I2.
:- value(X, Y1, I), value(X, Y2, I), Y1 != Y2.
:- value(X1, Y, I), value(X2, Y, I), X1 != X2.
:- samebox(X1, Y1, X2, Y2), value(X1, Y1, I), value(X2, Y2, I), X1 != X2, Y1 !=
Y2.

% OUTPUT
#show fixed/3.
```

Lab 4

```
%---------- THE TOWERS OF HANOI ----------%

% DISKS
#const m = 4.

% STEPS
#const n = 15.
```

```
% FACTS
peg(a; b; c).
disk(1..m).
at(1..m, a, 0).
goal(1..m, c).

% RULE
1 {move(D, P, T) : disk(D), peg(P)} 1 :- T = 0..n-1.

% EFFECT AXIOM
at(D, P, T + 1) :- move(D, P, T), T < n.

% FRAME AXIOM
at(D, P, T + 1) :- at(D, P, T), not move(D, _, T), T < n.

% INTEGRITY CONSTRAINTS
:- goal(D, P), not at(D, P, n).
:- at(D2, P2, T), move(D1, P2, T), D1 > D2, T < n.
:- at(D, P, T), move(D, P, T), T < n.
:- at(D1, P, T), at(D2, P, T), move(D1, _, T), D1 > D2, T < n.

% OUTPUT
#show move/3.
```

Lab 5

```
%---------- PIE TOWER ----------%

% PIES
#const m = 4.

% STEPS
#const n = 3.

% FACTS
on(1, 5, 0). on(4, 1, 0). on(2, 4, 0). on(3, 2, 0).
pie(1..m).
intermediateState(0..n-1).
state(0..n).

% RULE
1 { flip(X, T) : pie(X) } 1 :- intermediateState(T).

% GOAL
```

```
goal(X, Y) :- pie(X), pie(Y), Y = X + 1.

% AUXILARY PREDICATES
above(X, Y, T) :- on(X, Y, T), state(T).
above(X, Y, T) :- on(X, Z, T), above(Z, Y, T), state(T).
top(X, T) :- pie(X), state(T), not on(_, X, T).

% EFFECT AXIOMS
on(Y, X, T + 1) :- flip(Z, T), above(X, Z, T), on(X, Y, T),
intermediateState(T).
on(X, Y, T + 1) :- on(Z, Y, T), top(X, T), flip(Z, T), intermediateState(T).

% FRAME AXIOM
on(X, Y, T + 1) :- on(X, Y, T), flip(Z, T), not above(X, Z, T),
intermediateState(T).

% INTEGRITY CONSTRAINT
:- goal(X, Y), not on(X, Y, n).

% OUTPUT
#show flip/2.
```

Lab 6

```
%---------- MOVING ANIMALS FROM ONE CAGE TO ANOTHER ----------%

% STEPS
#const n = 3.

% FACTS
in(lion, a, 0).
in(tiger, b, 0).
in(sheep, c, 0).
in(cow, c, 0).
in(horse, d, 0).
carnivorous(lion; tiger).
hernivorous(sheep; cow; horse).
cage(a; b; c; d).

% RULES
animal(X) :- carnivorous(X).
animal(X) :- hernivorous(X).

% GOAL
```

```
goal :- in(tiger, c, n).
1 {move(A, C, T) : animal(A), cage(C)} 1 :- T = 0..n-1.

% EFFECT AXIOM
in(A, C, T + 1) :- move(A, C, T), T < n.

% FRAME AXIOM
in(A, C, T + 1) :- in(A, C, T), not move(A, C, T), T < n.

% INTEGRITY CONSTRAINTS
:- not goal.
:- in(A, C, T), in(B, C, T), carnivorous(A), hernivorous(B), T < n.

% OUTPUT
#show move/3.
```

Lab 7

```
%---------- COURSE SELECION ----------%

% FACTS
student(john). student(mary). student(helen).
course(db). course(ai1). course(ai2). course(thesis). course(java).
credits(db,6). credits(ai1,5). credits(ai2,5). credits(thesis,10).
credits(java,4).

% RULE
{select(X, Y) : course(Y)} :- student(X).

% INTEGRITY CONSTRAINTS
:- student(X), #sum {Z, Y : select(X, Y), credits(Y, Z)} < 10.
:- #count {X : select(X, thesis)} > 1.

% OUTUPUT
#show select/2.
```

```
%---------- TRAVELLING SALESMAN ----------%

% TRAVELLS
#const n = 4.
```

```
% FACTS
city(a). city(b). city(c). city(d).
road(a,b,10). road(b,c,20). road(c,d,25). road(d,a,40).
road(b,d,30). road(d,c,25). road(c,a,35).
at(a, 0).

% RULES
1 {go(X, T) : city(X)} 1 :- T = 0..n-2.
visited(C) :- at(C, T), T = 0..n-1.
go(a, n-1).

% MINIMIZATION
#minimize {D, X, Y, T : at(X, T), go(Y, T),road(X, Y, D)}.

% EFFECT AXIOM
at(Y, T+1):-at(X, T),road(X, Y, _), go(Y, T).

% INTEGRITY CONSTRAINTS
:- city(C), not visited(C).

% OUTPUT
#show go/2.
```

```
%---------- ROBOT LABYRINTH ----------%

% GRID
#const rows = 4.
#const cols = 4.
#const n = rows * cols.

% FACTS
blocked(2, 1).
blocked(2, 2).
blocked(3, 3).
direction(up; down; left; right).
robot(1, 1, 0).

% GOAL
goal:-robot(3, 2, n).

% RULES
0 {move(D, T) : direction(D)} 1 :- T = 0..n-1.
action(T) :- move(_, T).
lastAction(T) :- T = #max {X : action(X)}.
```

```

% MINIMIZATION
#minimize {T : lastAction(T)}.

% EFFECT AXIOMS
robot(R+1, C, T+1) :- robot(R,C,T), move(down,T), R < rows,  T < n.
robot(R-1, C, T+1) :- robot(R,C,T), move(up,T),  R > 1, T < n.
robot(R, C+1, T+1) :- robot(R,C,T), move(right,T), C < cols, T < n.
robot(R, C-1, T+1) :- robot(R,C,T), move(left,T), C > 1, T < n.

% FRAME AXIOM
robot(R, C, T+1) :- robot(R, C, T), not action(T), T < n.

% INTEGRITY CONTSTRAINTS
:- robot(R, C, T), blocked(R, C).
:- not goal.

% OUTPUT
#show move/2.
```

Lab 8

```
%---------- FARMER CROSSES RIVER ----------%

% STEPS
#const n=3.

% FACTS
object(wolf; sheep; hay).
position(wolf, left, 0).
position(sheep, left, 0).
position(hay, left, 0).
position(boat, left, 0).

% GOAL
goal :- position(wolf, right, n), position(sheep, right, n), position(hay, right,
n), position(boat, right, n).

% RULE
1 { move(X,T) : object(X) } 2 :- T = 0..n-1.

% EFFECT AXIOM
position(X, right, T+1) :- position(X, left, T), move(X, T), position(boat, left,
T), T < n.
```

```
position(X, left, T+1) :- position(X, right, T), move(X, T), position(boat,
right, T), T < n.
position(boat, left, T+1) :- position(boat, right, T), T < n.
position(boat, right, T+1) :- position(boat, left, T), T < n.

% FRAME AXIOM
position(X, Y, T+1) :- position(X, Y, T), not move(X, T), object(X), T < n.

% INTEGRITY CONSTRAINTS
:- not goal.
:- position(wolf, X, T), position(sheep, X, T), position(boat, Z, T), X!= Z.
:- position(sheep, X, T), position(hay, X, T), position(boat, Z, T), X!= Z.

% OUTPUT
#show move/2.
```

Lab 9

```
%---------- OPTIMAL PATH ----------%

% NODES
#const start = 1.
#const end = 5.

% FACTS
edge(1,2,3).
edge(2,3,2).
edge(3,4,3).
edge(4,1,7).
edge(1,4,7).
edge(4,3,3).
edge(3,5,10).
edge(4,5,4).
edge(5,2,6).

% RULES
{inPath(X,Y) : edge(X,Y,_)}.
reachable(X,Y) :- inPath(X,Y).
reachable(X,Y) :- reachable(X,Z),   inPath(Z,Y).

% OPTIMIZATION
```

```
#minimize{Z,X,Y: inPath(X,Y), edge(X,Y,Z)}.

% Integrity constraints
:- not reachable(start,end).

% OUTPUT
#show inPath/2.
```