Johdatus tekoälyyn, syksy 2011, Teemu Roos **Harjoitus 3**

Huom: Voit saada tästä harjoituskerrasta max. 6 pistettä.

Tehtävä 1. Looginen päättely (2 pistettä)

a) (1 piste).

- Lause $A \Rightarrow A \vee B$ (jos A on tosi, on myös A tai B tosi) on esimerkki tautologisesta lauseesta.
- Lause $(A \vee B) \wedge \neg A \wedge \neg B$ (eli "A tai B" ja ei-A ja ei-B) on esimerkki ristiriitaisesta lauseesta.
- Fermat's viimeinen teoreema sanoo, että lause

  $$a, b, c, \text{ ja } n \text{ ovat positiivisia kokonaislukuja, } n > 2, \text{ ja } a^n + b^n = c^n$$

  on ristiriitainen. Fermat väitti löytäneensä yksinkertaisen todistuksen teoreemalle vuonna 1637, mutta kesti satoja vuosia ennen kuin Andrew Wiles todisti sen vuonna 1995. (Wilesin todistus ei ole yksinkertainen.)
- Kysymys päteekö lause

  $$\text{NP} = \text{P},$$

  on ehkä tunnetuin tietojenkäsittelytieteen avoimista ongelmista (printtaamisen lisäksi). Kuten kaikki hyvin määritellyt väitteet, se on välttämättä joko tosi (tautologinen) tai epätosi (ristiriitainen). (Koska väitteessä ei esiinny "vapaita" muuttujia, kuten lauseessa $x > 2$ esiintyvä x, lause ei voi olla kontingentti.)

  Koska lauseen NP = P totuusarvoa ei osata ratkaista, on periaatteessa mahdollista, että se on ratkeamaton, ts. sitä ei *voi* todistaa todeksi tai epätodeksi. Huomaa että tästä huolimatta lause on joko tosi tai epätosi.

b) (1 piste). Kirjoita Prolog-ohjelma, joka esittää seuraavan joukon lauseita:
Sokrates omistaa Rekku-koiran.
Jos Sokrates omistaa X:n, X on Sokrateen.
Rekku-koira on Jekku-pennun isä.
Jos X on Y:n isä, X on isä.
Jos X on Sokrateen ja X on isä, X on Sokrateen isä.

```
% VERSIO 1
oma(sokrates,rekku).              % Sokrates omistaa Rekku-koiran
sokrateen(X):-oma(sokrates,X).    % jos Sokrates omistaa X:n, X on "Sokrateen"
isa(rekku,jekku).                 % Rekku-koira on Jekku-pennun isa
isa(X):-isa(X,Y).                 % Jos X on Y:n isa, X on isa
sokrateen_isa(Y):-sokrateen(Y),isa(Y).   % Jos X on Sokrateen ja X on isa,
                                         % X on Sokrateen isa
```

```
% VERSIO 2
oma(sokrates,rekku).              % Sokrates omistaa Rekku-koiran
isa(rekku,jekku).                 % Rekku-koira on Jekku-pennun isa
isa(X):-isa(X,Y).                 % Jos X on Y:n isä, X on isa
sokrateen_isa(Y):-isa(Y),oma(sokrates,Y).   % Jos X on Sokrateen ja X on isa,
                                            % X on Sokrateen isa
```

Kummalla tahansa versiolla, saadaan seuraava tulos.

```
$ prolog
GNU Prolog 1.3.0
By Daniel Diaz
Copyright (C) 1999-2007 Daniel Diaz
| ?- consult('sokrateen_isa').
compiling /home/ttonteri/teaching/ai/code/sokrateen_isa.pl for byte code...
/home/ttonteri/teaching/ai/code/sokrateen_isa.pl:4: warning: singleton variables [Y] for isa/1
/home/ttonteri/teaching/ai/code/sokrateen_isa.pl compiled, 6 lines read - 1023 bytes written, 10

yes
| ?- sokrateen_isa(X).

X = rekku

yes
```

Tässä vielä kolmas versio, joka osoittaa joitain prologin kummallisuuksia:

```
oma(sokrates,rekku).              % Sokrates omistaa Rekku-koiran
isa(rekku,jekku).                 % Rekku-koira on Jekku-pennun isa
isa(X,Y):-oma(Y,X),isa(X).        % Jos Y omistaa X:n ja X on isa,
                                  % X on Y:n isä isa(X):-isa(X,Y).
isa(X):-isa(X,Y).                 % Jos X on Y:n isä, X on isä
```

(Erikoisuus numero 1: Molemmat `ISA(.,.)` lauseet on sijoitettava peräkkäin, ennen lausetta `ISA(X):-isa(X,Y).`, koska jälkimmäisessä seurauksena oleva `ISA(.)`-predikaatti tulkitaan eri predikaatitksi kuin `ISA(.,.)`, ja lauseiden, joiden seurauksena on tietty predikaatti, on esiinnyttävä peräkkäin.)

Kun ajetaan tämä ohjelma, saadaan seuraava tulos.

```
$ prolog
GNU Prolog 1.3.0
By Daniel Diaz
Copyright (C) 1999-2007 Daniel Diaz
| ?- consult('sokrateen_isa').
compiling /home/ttonteri/teaching/ai/code/sokrateen_isa.pl for byte code...
/home/ttonteri/teaching/ai/code/sokrateen_isa.pl:4: warning: singleton variables [Y] for isa/1
/home/ttonteri/teaching/ai/code/sokrateen_isa.pl compiled, 6 lines read - 1023 bytes written, 10

yes
| ?- isa(X,sokrates).
```

```
X = rekku ? ;

X = rekku ? ;

X = rekku ? ;
```

(Erikoisuus numero 2: Painamalla ';' uudelleen ja uudelleen, tulee aina uudeksi ratkaisuksi `X = rekku`. Prolog-tulkki siis tarjoaa kysymykseen loputtomasti samaa vastausta.)

Tehtävä 2. Shakki. (1-2 pistettä)

a) (1 piste). Toteuta valmiina annettuun runkoon heuristinen evaluointifunktio (luokka `YourEvaluator`), joka arvioi shakkilaudan tilanteen. Funktion tulee palauttaa sitä suurempi arvo, mitä luultavammin peli päättyy valkean voittoon.

Luokan `OurEvaluator` (a.k.a. Deep Glue) lähdekoodi on esitetty kuvassa 1.

b) (1 piste).

Katso `http://tmtynkky.users.cs.helsinki.fi`.

Rankkaus on toteutettu ns. Elo-rankingin avulla (ks. luennon 8 luentokalvojen toiseksi viimeinen sivu tai `http://en.wikipedia.org/wiki/Elo_rating_system`.)

```
public class OurEvaluator extends Evaluator {
  static double blackBoostTable [][] = {
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0.15,0.2,0.2,0.15,0,0},
    {0,0,0.2,0.3,0.3,0.2,0,0},
    {0,0,0.15,0.1,0.1,0.15,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0}
  };
  static double whiteBoostTable [][] = {
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0.15,0.1,0.1,0.15,0,0},
    {0,0,0.2,0.3,0.3,0.2,0,0},
    {0,0,0.15,0.2,0.2,0.15,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0}
  };
  public double eval(Position p) {
    int offsetx = 0; int offsety = 0;
    if (p.bCols == 6) offsetx = 1;
    if (p.bRows == 6) offsety = 1;
    double ret = (Math.random()-.5);
    for(int x = 0; x < p.board.length; ++x) {
      for(int y = 0; y < p.board[x].length; ++y) {
        if(p.board[x][y] == Position.Empty) continue;
        if(p.board[x][y] == Position.WKing) ret += 1e9;
        if(p.board[x][y] == Position.WQueen) ret += 9;
        if(p.board[x][y] == Position.WRook) ret += 5.25;
        if(p.board[x][y] == Position.WBishop) ret += 3.25;
        if(p.board[x][y] == Position.WKnight) ret += 3;
        if(p.board[x][y] == Position.WPawn) ret += 1;
        if(p.board[x][y] == Position.BKing) ret -= 1e9;
        if(p.board[x][y] == Position.BQueen) ret -= 8.5-.05*y;
        if(p.board[x][y] == Position.BRook) ret -= 4.75-.05*y;
        if(p.board[x][y] == Position.BBishop) ret -= 2.75-.05*y;
        if(p.board[x][y] == Position.BKnight) ret -= 2.5-.05*y;
        if(p.board[x][y] == Position.BPawn) ret -= 1.-.1*y;

        if(Position.isWhitePiece(p.board[x][y])) {
          ret += whiteBoostTable[y+offsety][x+offsetx];
        } else if(Position.isBlackPiece(p.board[x][y])) {
          ret -= blackBoostTable[y+offsety][x+offsetx];
        }
      }
    }
    return ret;
  }
}
```

Figure 1: `OurEvaluator.java`

Tehtävä 3. Probabilistinen päättely (1-2 pistettä)

a) (1 piste).

Autotehtävä:

```
#!/usr/bin/python
import random

pA = 0.9
pRA = 0.9
pSA = 0.95
pB = 0.95
pKAB = 0.99
pLK = 0.99
N = 100000

class Monikko:
    [A,R,S,B,K,L] = [False,False,False,False,False,False]

kasa=[]
for i in range(N):
    m = Monikko()
    if random.random() < pA: m.A = True
    if m.A and random.random() < pRA: m.R = True
    if m.A and random.random() < pSA: m.S = True
    if random.random() < pB: m.B = True
    if m.A and m.B and random.random() < pKAB: m.K = True
    if m.K and random.random() < pLK: m.L = True
    kasa.append(m)

cnotL = len([m for m in kasa if not m.L])
cnotAnotL = len([m for m in kasa if not m.A and not m.L])
cnotBnotL = len([m for m in kasa if not m.B and not m.L])

print 'P[not A | not L]: %.1f%%' % (100.*cnotAnotL/cnotL)
print 'P[not B | not L]: %.1f%%' % (100.*cnotBnotL/cnotL)

cnotLnotR = len([m for m in kasa if not m.L and not m.R])
cnotAnotLnotR = len([m for m in kasa if not m.A and not m.L and not m.R])
cnotBnotLnotR = len([m for m in kasa if not m.B and not m.L and not m.R])

print 'P[not A | not L, not R]: %.1f%%' % (100.*cnotAnotLnotR/cnotLnotR)
print 'P[not B | not L, not R]: %.1f%%' % (100.*cnotBnotLnotR/cnotLnotR)
```

(Esimerkki valitettavasti vain pythoniksi. Toivottavasti osaat tulkita sitä riittävän hyvin.)

Generoi näin 100000 kokonaista monikkoa. Laske sen jälkeen kuinka suuressa osassa monikoista, joissa $L = 0$, pätee $A = 0$. Entä kuinka suuressa osassa monikoista, joissa $L = 0$, pätee $B = 0$? Esitä tulkinta saamillesi tuloksille.

Laske myös kuinka suuressa osassa generoimiasi monikoita, joissa pätee $L = 0$ ja $R = 0$, pätee $A = 0$. Entä kuinka suuressa omassa monikoista, joissa pätee $L = 0$ ja $R = 0$, pätee $B = 0$? Vertaa näitä tuloksia edellä saamiisi ja esitä tulkinta havainnoillesi.

```
$ ./auto.py
P[not A | not L]: 61.7%
P[not B | not L]: 31.1%
P[not A | not L, not R]: 94.2%
P[not B | not L, not R]: 9.2%
```

Tulkinta: Jos auto ei liiku, on n. 61.7% todennäköisyydellä akussa vikaa ja n. 31.1% todennäköisyydellä bensatankki tyhjä. Jos taas tiedetään, että radio ei soi, ovat ko. todennäköisyydet n. 94.2% ja n. 9.2%. Radion toimimattomuus siis viittaa akkuvikaan, jolloin ei ole syytä epäillä että bensa olisi lopussa.

b) (1 piste).
Hälytintehtävä:

```
#!/usr/bin/python
import random

pM = 0.014
pR = 0.0062
pHnotMnotR = 0.039
pHMnotR = 0.84
pHnotMR = 0.918
pHMR = 0.974
N = 100000

class Monikko:
    [M,R,H] = [False,False,False]

kasa=[]
for i in range(N):
    m = Monikko()
    if random.random() < pM: m.M = True
    if random.random() < pR: m.R = True
    if not m.M and not m.R and random.random() < pHnotMnotR: m.H = True
    if m.M and not m.R and random.random() < pHMnotR: m.H = True
    if not m.M and m.R and random.random() < pHnotMR: m.H = True
    if m.M and m.R and random.random() < pHMR: m.H = True
    kasa.append(m)

cH = len([m for m in kasa if m.H])
cRH = len([m for m in kasa if m.R and m.H])
cMRH = len([m for m in kasa if m.M and m.R and m.H])

print 'P[R | H]: %.1f%%' % (100.*cRH/cH)
print 'P[R | M,H]: %.1f%%' % (100.*cMRH/cRH)
```

Generoi sitten 100000 kokonaista kolmikkoa. Laske kuinka suuressa osassa monikkoja, joissa $H = 1$, pätee $R = 1$. Laske myös kuinka suuressa osassa monikkoja, joissa pätee sekä $H = 1$ että $M = 1$, pätee lisäksi $R = 1$.

Esitä tulkinta havainnoillesi. Kumpi edellä mainituista osuuksista on suurempi? Osaatko sanoa mitä se merkitsee?

Ajetaan esimerkkiajo:

```
$ ./earthquake.py
P[R | H]: 10.3%
P[R | M,H]: 2.5%
```

Tulkinta: Kun hälytin soi, on ryöstön todennäköisyys n. 10.3%. Jos lisäksi tiedetään, että alueella on ollut maanjäristys, laskee ryöstön todennäköisyys n. 2.5%:iin. Kyseessä on ns. “explaining away”-ilmiö, eli havainnoille löytyy vaihtoehtoinen selitys (maanjäristys), jonka takia muita selityksiä ei tarvita.