Programare Logică Probabilistă

X

October 2018

Cuprins

1 Introducere

2 Noțiuni teoretice

2.1 Teoria elementară a probabilităților

În acest subcapitol vom acoperi concepte și noțiuni de bază din teoria probabilităților precum operații cu evenimente și formule de calcul pentru probabilitățile acestora, conceptul de probabilitate condiționată, evenimente independente etc.

Notații.

Ω - spațiul probelor ce reprezintă mulțimea tuturor rezultatelor posibile într-un context dat

$\emptyset$ - evenimentul imposibil

$\mathcal{P}(\Omega) = \{A \mid A \subseteq \Omega\}$ - mulțimea submulțimilor de evenimente posibile din spațiul probelor Ω

Definiție 2.1.1. Fie Ω multime. $\mathcal{F} \subseteq \mathcal{P}(\Omega)$ se numește *corp borelian* sau *σ-algebra* $\iff$ urmatoarele afirmații sunt adevarate:

1. $\mathcal{F} \neq \emptyset$
2. $A \in \mathcal{F} \implies \overline{A} \in \mathcal{F}$, unde $\overline{A}$ complementul mulțimii A.
3. $A_1, A_2, ..., A_n \in \mathcal{F} \implies \bigcup_n A_n \in \mathcal{F}$

$(\Omega, \mathcal{F})$ se numește spațiu măsurabil când K este corp borelian pe Ω.

Definiție 2.1.2. Fie $(\Omega, \mathcal{F})$ spațiu măsurabil. Se numește *funcție de probabilitate* orice funcție $\Pr : \mathcal{F} \to \mathbb{R}$ ce satisface următoarele condiții:

1. $\forall A \in \mathcal{F}$, $Pr(A) \geqslant 0$ (nenegativă)
2. $\Pr(\Omega) = 1$
3. pentru orice secvență finită sau numărabilă de evenimente reciproc exclusive $A_1, A_2, ..A_n \in \mathcal{F}$ (i.e. $\forall A_i, A_j \in \mathcal{F}, A_i \cap A_j = \emptyset$), $Pr(\bigcup_{i=1}^{n} A_i) = \sum_{i=1}^{n} Pr(A_i)$

$(\Omega, \mathcal{F}, Pr)$ se numește *câmp de probabilitate* $\iff$ Pr este funcție de probabilitate pe spațiul măsurabil $(\Omega, \mathcal{F})$.

În continuare pentru următoarele noțiuni vom considera câmpul de probabilitate $(\Omega, \mathcal{F}, Pr)$.

Lemă 2.1.3. Oricare două evenimente $E_1, E_2 \in \mathcal{F}$.

$$Pr(E_1 \cup E_2) = Pr(E_1) + Pr(E_2) - Pr(E_1 \cap E_2)$$

Definiție 2.1.4. Două evenimente $E_1, E_2 \in \mathcal{F}$ sunt *independente* ddacă:

$$Pr(E_1 \cap E_2) = Pr(E_1) \cdot Pr(E_2)$$

Generalizând, fie $E_1, E_2, ..., E_n \in \mathcal{F}$ evenimente, acestea sunt reciproc independente ddacă: $Pr(\bigcap_{i=1}^{n} E_i) = \prod_{i=1}^{n} Pr(E_i)$

Observație 1. Dacă $E_1, E_2 \in \mathcal{F}$ sunt evenimente independente au loc egalitățile:

$Pr(A \cap \overline{\mathrm{B}}) = Pr(A) \cdot Pr(\overline{\mathrm{B}})$

$Pr(\overline{\mathrm{A}} \cap B) = Pr(\overline{\mathrm{A}}) \cdot Pr(B)$

$Pr(\overline{\mathrm{A}} \cap \overline{\mathrm{B}}) = Pr(\overline{\mathrm{A}}) \cdot Pr(\overline{\mathrm{B}})$

Definiție 2.1.5. *Probabilitatea condiționată* ca evenimentul E_1 să aibă loc știind că evenimentul E_2 se întâmplă are următoarea reprezentare:

$$Pr(E_1 \mid E_2) = \frac{Pr(E_1 \cap E_2)}{Pr(E_2)}$$

Probabilitatea condiționată este bine definită dacă $Pr(E_2) \neq 0$

Definiție 2.1.6. Fie $B_i \subseteq \Omega, \forall i \in I$. $(\mathrm{B_i})_{i \in I}$ se numește *partiție* a lui Ω dacă și numai dacă $(\mathrm{B_i})_{i \in I}$ sunt disjuncte două câte două (i.e. $\forall i, j \in I, B_i \cap B_j = \emptyset$) și $\bigcup_{i \in I} B_i = \Omega$.

Teoremă 1. (*Teorema probabilității totale*). Fie câmpul de probabilitate $(\Omega, \mathcal{F}, Pr)$ și $(\mathrm{B_i})_{i \in I} \subset \mathcal{F}$ partiție cel mult numărabilă a lui Ω astfel încât $Pr(B_i) > 0$, $\forall i \in I$. Atunci, $\forall A \in \mathcal{F}$:

$$P(A) = \sum_{i \in I} P(A \mid B_i) \cdot P(B_i)$$

Teoremă 2. (*Teorema lui Bayes*) Fie câmpul de probabilitate $(\Omega, \mathcal{F}, Pr)$ si $E_1, E_2 \in \mathcal{F}$ astfel încât $Pr(E_1) \neq 0$ și $Pr(E_2) \neq 0$. Atunci:

$$Pr(E_2 \mid E_1) = \frac{Pr(E_1 | E_2)}{Pr(E_1)}$$

2.2 Noțiuni teoretice ale programării logice

2.2.1 Rezoluția SLD

SLD (Selective, Linear, Definite clause resolution) este regula de bază pentru inferență întâlnită în programarea logică, implicit și în cazul limbajului Prolog. Astfel, fie T

o mulțime de clauze definite, regula rezoluției SLD este următoarea:

1. dată fiind clauza țintă: $\neg L_1 \vee ... \vee \neg L_i \vee ... \neg L_n$,
2. având literalul L_i și o clauză definită $L \vee \neg Q_1 \vee ... \vee \neg Q_k$,
3. obținem $(\neg L_1 \vee ... \vee \neg Q_1 \vee ... \vee Q_K \vee ... \vee L_n)\theta$,

unde $L \vee \neg Q_1 \vee ... \vee \neg Q_k \in$ T (în care toate variabilele au fost redenumite) și θ este cel mai general unificator pentru L_i și L [5].

Teoremă 3. *(Completitudinea rezoluției SLD).* Sunt echivalente următoarele:

- există o SLD-respingere a lui $P_1 \wedge ... \wedge P_m$ din T,
- T $\models P_1 \wedge ... \wedge P_m$.

Exemplu 1. Gasiți o SLD-respingere.

1. r :- p, q.
2. s :- p, q.
3. v :- t, u.
4. w :- v, s.
5. t.
6. q.
7. u.
8. p.

? - w.

Pasul 1: $G_0 = \neg$ w
Pasul 2: $G_1 = \neg$ v $\vee \neg$ s (aplicăm rezoluția SLD pentru G_0 și clauza 4)
Pasul 3: $G_2 = \neg$ t $\vee \neg$ u $\vee \neg$ s (aplicăm rezoluția SLD pentru G_1 și clauza 3)
Pasul 4: $G_3 = \neg$ u $\vee \neg$ s (din clauza 5)
Pasul 5: $G_4 = \neg$ s (din clauza 7)
Pasul 6: $G_5 = \neg$ p $\vee \neg$ q (aplicând rezoluția SLD pentru G_5 și clauza 2)
Pasul 7: $G_6 = \neg$ p (din clauza 6)
Pasul 8: $G_7 = \square$ (din clauza 8)

2.2.2 Diagrame binare de decizie (BDDs)

Pentru o mai ușoară definire a diagramelor binare de decizie vom începe prin a prezenta termenul de arbori binari de decizie.

Conceptual, **arborii binari de decizie** păstrează ideea din spatele arborilor binari ordinari, dar se deosebesc de aceștia întrucât modeleaza formule boolene. Presupunând că avem variabilele $x_1, x_2, ..., x_n$ ale unei formule boolene, putem construi arborele binar de decizie alegând aleator un nod rădacină, să spunem x_1 ce va avea în descendență doi subarbori, unul pentru $x_1 = 1$ și unul pentru $x_2 = 0$. Continuând în aceeași manieră, vom lua următoarele variabile din formula booleana și vom construi subarbori până ce în final vom ajunge la nodurile frunză ce reprezintă valoarea finală a formulei. Pentru fiecare nod frunză în parte, urmând drumul către rădăcină vom obține de pe muchiile drumului valoarea asignată fiecărei variabile ce a dus la rezultatul frunzei. [6]

Exemplu 2. Considerăm formula: $f(x_1, x_2) = x_1 \wedge x_2, f : \{0,1\}^2 \mapsto \{0,1\}$

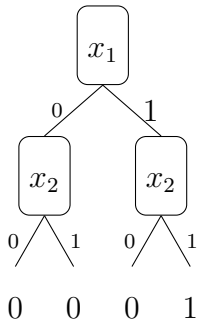

Figura 1. Arborele binar de decizie asociat formulei boolene

Totuși, arborii binari de decizie întâmpină doua mari probleme. Pe de-o parte, pentru un număr dat de n variabile, arborele va ajunge să aibă $2^{(n-1)}$ noduri de decizie și încă 2^n frunze pe ultimul nivel. Pe de alta parte, arborii binari de decizie mențin subarbori nefolositori, precum ramura stângă a nodului rădăcină din exemplul de mai sus, întrucât pentru $x_1 = 0$, rezultatul funcției va fi, indiferent de x_2, 0.

Din aceste motive, ajungem la varianta îmbunătățită a acestor arbori binari de decizie prin intermediul **diagramelor binare de decizie**. Acestea permit omiterea cazurilor redundante, ce nu influențează în niciun fel rezultatul funcției pe acea ramura, și, de asemenea, acceptă refolosirea nodurilor de catre subarbori identici. În

teorie, o diagramă binară de decizie poate avea un număr exponențial de noduri, dar în majoritatea cazurilor din practică s-a observat o îmbunătățire majoră datorită folosirii acestei structuri în defavoarea arborilor binari de decizie.

3 Problog

3.1 Aspecte generale

Problog, extensia probabilistă a Prolog, reprezintă o componentă software ce are la bază limbajul de programare Python. Acesta este un limbaj generic cu utilități în multiple arii precum *Dezvoltare Web*, *Învățare Automată*, *Statistică*. Un program în Problog definește o distribuție asupra programelor logice, întrucât specifică fiecărei clauze în parte probabilitatea de a aparține unui exemplu ales aleator (în cazul acesta, un exemplu semnifică un program logic non-probabilistic în care fiecărei clauze i s-a atribuit o valoare de adevăr pe baza probabilității din programul inițial). Acest atribut reprezintă noutatea adusă limbajului Prolog. Spre exemplu, dacă vrem să aflăm probabilitatea ca o casă să fie dărâmată din cauza unei calamități naturale am putea scrie următorul cod:

```
0.4::cutremur.
0.2::tsunami.
0.1::uragan.

casaDaramata :- cutremur.
casaDaramata :- tsunami.
casaDaramata :- uragan.
```

În codul de mai sus am atribuit fiecărei fapte câte o probabilitate ca aceasta să fie adevarată după care am descris predicatul "casaDaramata". Astfel, dacă ulterior am vrea să răspundem la întrebarea `query(casaDaramata).` am primi răspunsul `casaDaramata: 0.568` . Răspunsul reprezintă probabilitatea ca întrebarea pe care am adresat-o să fie adevarată, luând în considerare cele trei fapte.
ProbLog verifică probabilitatea primei reguli din program `casaDaramata :- cutre`

`mur.` care este 0.4, după care trecând mai departe la cea de-a doua regulă caută probabilitatea de a se produce un tsunami, fără a exista un cutremur, iar cea din urmă regulă ia în calcul variantele în care are loc un uragan, fără a exista nici un cutremur și nici un tsunami. În final adună cele trei probabilități reieșite și ajunge la răspunsul final pentru întrebarea adresată în cazul acesta.

3.1.1 Semantica Problog

Un program în Problog este un set de clauze definite de forma $p_i : c_i$, unde p_i reprezintă probabilitatea, iar c_i o clauză definită asemeni unei implicații de forma $u <- p \wedge q \wedge .. \wedge t$. De asemenea, un aspect important este faptul că fiecare clauză este reciproc independentă de oricare alta. Așadar, un program în Problog poate fi scris sub forma:

$$T = \{p_1 : c_1, p_2 : c_2, ..., p_n : c_n\}$$

Mulțimea T definește o distribuție de probabilitate peste mulțimea programelor logice L ($L \subseteq L_T$, unde $L_T = \{c_1, c_2, ..., c_n\}$), astfel:

$$Pr(L|T) = \prod_{c_i \in L} p_i \cdot \prod_{c_i \in L_T \setminus L} (1 - p_i)$$

Formula de mai sus poate fi citită în modul următor: valoarea de probabilitate astfel încât clauzele din mulțimea L să fie adevarate, iar restul clauzelor programului logic ce nu fac parte din submulțimea L a lui L_T să fie false.

În continuare, putem extinde estimarea probabilității la cea a unei întrebări adresate unui program logic in Problog. O întrebare în cazul de față este reprezentată de un predicat căruia vrem să-i aflăm probabilitatea de a fi adevarat. Considerând întrebarea q peste programul logic T, putem calcula probabilitatea de succes $Pr(q|T)$ în modul următor:

$$Pr(q|L) = \begin{cases} 1, & \exists\theta : L \models q\theta \\ 0, & altfel \end{cases}.$$

$$Pr(q, L|T) = Pr(q|L) \cdot Pr(L|T)$$

$$Pr(q|T) = \sum_{M \subseteq L_T} Pr(q, M|T), \qquad [7]$$

unde L este o submulțime a lui L_T (mulțimea tuturor clauzelor definite ale programului logic T). În limbaj natural, putem spune că probabilitatea de succes a unei întrebări q corespunde probabilității ca aceasta să aibă o demonstrație, dată fiind distribuția de probabilitate peste mulțimea programelor logice ale lui T.

3.1.2 Aspecte teoretice

Contribuția de seamă a acestei extensii adusă limbajului Prolog este reprezentată printr-un nou mod concret de rezolvare pentru calculul probabilității de succes a unei întrebări. Pe scurt, ideea din spatele acestui tool se bazeaza pe rezoluții SLD și metode de calcul a probabilității formulelor booleene, iar algoritmul propus de aproximare a probabilităților este construit ca fiind o combinație între un DFS iterativ (Depth-First-Search) și diagrame binare de decizie, dat fiind recentul progres în aceasta direcție.

Una dintre principalele motivații din spatele acestei extensii a Prolog stă în spatele necesității de a găsi soluții la probleme din viața reală precum cele din domeniul biologiei. Acestea necesita deseori un timp îndelungat de căutare a soluțiilor, spre exemplu: determinarea legăturii unei gene cu o anumită afecțiune. Problog modelează perfect acest tip de probleme deoarece putem face urmatoarele asocieri:

unde fiecare nod în parte reprezintă fie o componentă genetică, fie o afecțiune ce poate rezulta în boala țintă (echivalentul unei întrebări în Problog), iar muchia orientată $x \mapsto y$ reprezintă probabilitatea ca, având îndeplinită condiția x, să putem obține afecțiunea y. Graful asociat obținerii unei probabilități pentru o întrebare este orientat, aciclic și neconex.

3.1.3 Comparație cu alte framework-uri

În ultimele două decade s-au dezvoltat multe alte framework-uri dedicate programării logice probabiliste precum: PHA[Poole, 1993], PRISM[Sato and Kameya, 2001],

SLPs[Muggleton, 1996], probabilistic Datalog (pD) [Fuhr, 2000]. Toate aceste frameworkuri atașează probabilități formulelor logice, de obicei clauzelor definite, dar în majoritatea cazurilor fiecare impune anumite restricții. Spre exemplu, în SLP clauzele ce definesc același predicat nu pot fi considerate simultan adevarate. PRISM și PHA de asemenea nu suportă ca anumite clauze să aibă loc simultan, iar toate aceste restricții simplifică atât calculul probabilității unei întrebări, cât și algoritmul pe baza căruia se fac deducțiile.

Totuși, un framework vag asemănător Problog-ului este pD întrucât, spre deosebire de cele anterior menționate nu impune restricții asupra combinațiilor posibile de clauze ce pot fi simultan adevarate. De asemenea, atât Problog, cât și pD se deosebesc de restul framework-urilor prin faptul că probabilitatea fiecărei clauze este independentă față de oricare alta. Cu toate acestea, un mare dezavantaj al pD este mecanismul de deducție utilizat. Acesta este unul naiv ce poate evalua cel mult 10 conjuncții. În opoziție, algoritmul de aproximare utilizat de Problog este capabil să furnizeze un rezultat pentru până la 10000 de formule. [7]

3.2 Instalare

Pentru a putea folosi toate utilitățile pe care Problog le pune la dispoziție avem nevoie de o versiune Python 2.7 sau 3 deoarece acesta vine totodata cu întreaga suită de pachete oferite de PyPI (Python Package Index - site-ul oficial afiliat acestui limbaj de programare).

Odată dobândită această cerință minimală putem utiliza Problog atât din consolă, cât și dintr-un editor online. Dacă alegem să folosim Problog din consolă o putem face prin intermediul comenzii `problog shell` ce va deschide o interfață asemănătoare aceleia pentru Prolog prin intermediul căreia putem da comenzi similare. [2]

```
% Welcome to ProbLog 2.1 (version 2.1.0.34)
% Type 'help.' for more information.
?- consult('test.pl').
?- ordonat([1,2,3]).
True
```

```
?-
```

3.3 Utilizări din linia de comandă

Anterior am prezentat cum ne putem folosi de opțiunea `problog shell` pentru a putea utiliza Problog într-o manieră asemănătoare Prolog. Însă, Problog pune la dispoziție mai multe opțiuni de utilizare precum `sample`, `mpe`, `lfi`, `explain` ce pot fi adăugate cuvântului principal `problog` din linia de comandă.

Prima variantă în care putem folosi Problog este cea **default** pe care o putem apela din linia de comandă printr-o sintaxă de felul `problog numeFisier.pl`. Presupunând că fișierul pe care l-am dat conține și o serie de întrebari, comanda rulată va afișa probabilitatea asociată calculată pentru fiecare întrebare în parte. Spre exemplu, dat fiind următorul cod:

```
0.5::heads1.
0.6::heads2.
someHeads :- heads1.
someHeads :- heads2.

query(someHeads).
```

și apelând `problog someHeads.pl`, se va afișa `someHeads: 0.8` având semnificația că predicatul someHeads despre care am intrebat are o probabilitate de 0.8 să fie adevarată. Totuși, dacă dorim să extindem răspunsul putem apela `problog someHeads.pl -v`, unde `-v` provine de la cuvantul `verbose` (i.e. exprimat în multe cuvinte) și vom primi următorul rezultat:

```
[INFO] Output level: INFO
[INFO] Propagating evidence: 0.0000s
[INFO] Grounding: 0.0010s
[INFO] Cycle breaking: 0.0001s
[INFO] Clark‘s completion: 0.0000s
[INFO] DSharp compilation: 0.0061s
```

```
[INFO] Total time: 0.0092s
someHeads: 0.8
```

Cu ajutorul acestei comenzi observăm toți pașii din spatele rezolvării unei întrebări. Problog primește un model sub forma programului dat și calculează probabilitățile întrebărilor, având la bază transformarea codului inițial într-o reprezentare a unei baze de cunoaștere. Ulterior, codul transformat în formatul dorit poate fi rezolvat cu ajutorul unui compilator de cunoaștere asociat bazei de cunoaștere pe care am folosit-o. Toate aceste compilatoare de cunoaștere se preocupă de convertirea unor forme generale de "cunoaștere" în forme mai usor de evaluat.

Problog pune la dispoziție următoarele reprezentări ale unor baze de cunoaștere propoziționale: SDD(Sentential Decision Diagram) și d-DNNF(Deterministic Decomposable Negation Normal Form).Pentru SDD avem asociat compilatorul de cunoaștere cu același nume, iar pentru d-DNNF avem asociate două posibile compilatoare c2d (sistem ce compileaza forma normală conjunctivă în d-DNNF) și dsharp(de asemenea compilează forma normală conjunctivă în d-DNNF, bazându-se pe Sharp-SAT).

Cea de-a doua variantă în care putem folosi Problog este cu ajutorul cuvântului cheie **sample** urmat de numărul de exemple pe care ni-l dorim. Apelând o comandă de felul `problog sample numeFisier.pl -N 10`, rezultatul asociat este o serie de zece exemple, fiecare fiind alcătuit dintr-un subset de fapte cărora li se atribuie o valoare de adevar în concordanță cu probabilitatea pe care o au, și predicatele calculate în concordanță cu faptele din care sunt alcatuite. Continuând cu ajutorul codului prezentat anterior, dacă am apela `problog sample someHeads.pl` vom obține următoarea ieșire:

```
----------------
someHeads.
----------------
```

```
someHeads.
----------------
someHeads.
```

După cum se poate observa, ulterior unui apel fără alți parametri, comanda **sample** nu furnizează prea multă informație, ci doar afișeaza întrebările adevarate în urma setului de fapte ales asociat fiecărui exemplu. Totuși, putem îmbunătăți ieșirea acestei comenzi adăugând mai multe opțiuni precum: `--with-probability`, `--with-facts`. Așadar, păstrând același exemplu și executând comanda `problog sample someHeads.pl -N 2 --with-facts --with-probability` obținem următoarea ieșire:

```
someHeads.
\+heads1.
heads2.
% Probability: 0.3
----------------
\+heads1.
\+heads2.
% Probability: 0.2
```

De data aceasta, spre deosebire de prima data, putem observa valoarea de adevăr a faptelor ce au dus la rezultatul predicatului `someHeads` și de asemenea putem vedea probabilitatea ca acel subset de fapte să fie ales. În primul exemplu ales cunoaștem că fapta `heads1` nu este adevarată, `heads2` este adevarată, iar acest lucru duce la îndeplinirea regulii `someHeads :- heads2`. De asemenea, la finalul exemplului este calculată probabilitatea ca valorile de adevar să fie asociate faptelor în modul respectiv (în cazul de față: `\+heads1` are probabilitatea 0.5, `heads2` are probabilitatea 0.6, deci probabilitatea totală a subsetului ales pentru primul exemplu este 0.5 * 0.6 = 0.3).

O altă modalitate în care putem utiliza cuvântul cheie `sample` este cu scopul de a estima probabilitatea întrebărilor pe baza unui set de exemple. Apelând comanda `problog sample someHeads.pl -N 20 --estimate` se vor genera 20 de exemple

aleatoare pe baza cărora se va calcula probabilitatea ca întrebarea someHeads să fie adevărată.

```
$ problog sample some_heads.pl -N 20 --estimate
% Probability estimate after 20 samples (334.2541 samples/second):
someHeads: 0.65
$ problog sample some_heads.pl -N 20 --estimate
% Probability estimate after 20 samples (316.8490 samples/second):
someHeads: 0.9
$ problog sample some_heads.pl -N 200 --estimate
% Probability estimate after 200 samples (332.2214 samples/second):
someHeads: 0.78
$ problog sample some_heads.pl -N 200 --estimate
% Probability estimate after 200 samples (332.9496 samples/second):
someHeads: 0.83
```

În exemplele de mai sus avem două apeluri în care numărul de exemple este 20 și două apeluri în care numărul de exemple este 200. Astfel, putem observa că un numar mai mic de exemple duce la o variație a răspunsului mai mare, pe când un număr ridicat de exemple pe baza caruia se trage o concluzie diferă mult mai puțin de la un apel la altul, iar răspunsul ia valori într-un interval mai restrans.

Alte modalități pentru utilizarea Problog este alături de cuvântul cheie `mpe` (Most Probable Explanation) sau alături de cuvântul cheie `lfi` (Learning from interpretations). Prima variantă, `problog mpe numeFisier.pl` va produce un set de fapte care respectă toate regulile date în numeFisier. Anume, pe lângă faptele anotate cu o anumită probabilitate, putem avea și niste fapte a căror valoare de adevăr o știm cu siguranță, iar acestea sunt reprezentate în program sub forma `evidence(fapt1, true)` sau `evidence(fapt1, false)`. În momentul în care apelăm comanda anterioară, Problog va încerca să găsească un set de atribuiri ale faptelor ce respectă toate structurile `evidence`. Cea de-a doua variantă anterior menționată este `problog lfi numeFisierDeInvatat.pl numeFisierCuDate.pl numeFisierModelInvatat.pl`. În cazul acesta, numeFisierDeInvatat va conține atât fapte, cât

și reguli, dar acum faptele nu vor mai avea probabilitatea cunoscută, ci se dorește să fie învățată pe baza exemplelor existente în numeFisierCuDate.pl. Astfel, pentru a denumi un fapt a cărui probabilitate trebuie învățată putem scrie `t(_) :: fapt`, iar valoarea inițială va fi aleasă aleator. Totuși, dacă dorim să pornim învățarea de la o anumită valoare, putem specifica între paranteze în locul caracterului `_`.

În final, o ultimă funcționalitate pe care o putem folosi din linia de comandă este `explain`. Acest cuvânt cheie deservește cu scopul de a oferi mai multe detalii despre modul în care se calculează probabilitățile întrebărilor, arătând în mod explicit probabilitatea fiecărei reguli de-a lungul programului și faptele luate în considerare. Cu toate acestea, în momentul actual, `explain` nu poate fi utilizat pentru programe ce conțin și instrucțiunea `evidence`. Pentru a apela la această funcționalitate putem scrie în linia de comanda `problog explain numeFisier.pl`. Astfel, pentru exemplul folosit până acum, comanda aceasta ar afișa următoarea secvență:

```
Transformed program
-------------------
0.5::heads1.
0.6::heads2.
someHeads :- heads1.
someHeads :- heads2.
query(someHeads).

Proofs
------
someHeads :- heads2. % P=0.6
someHeads :- heads1, \+heads2. % P=0.2


Probabilities
-------------
someHeads: 0.8
```

Aici putem observa toate cele trei părți prin care se trece în momentul în care utilizăm funcționalitatea de `explain`. Astfel, în prima fază programul este rescris, după care pentru fiecare întrebare în parte se va calcula o serie de clauze disjuncte a caror sumă va fi ulterior calculată drept răspuns al întrebării respective.

Bibliografie

[1] Introduction. - ProbLog: Probabilistic Programming,
`https://dtai.cs.kuleuven.be/problog/`

[2] ProbLog 2.1 documentation,
`https://problog.readthedocs.io/en/latest/`

[3] Cristian Niculescu. *Probabilități și statistică*. (Română) Editura Universității din București, 2014.

[4] Michael Mitzenmacher, Eli Upfal. *Probability and Computing - Randomized Algorithms and Probabilistic Analysis*. (Engleză) Cambridge University Press, 2005.

[5] Denisa Diaconescu, Ioana Leuștean. *Curs - Programare Logică*.
`https://cs.unibuc.ro/ ddiaconescu/2018/pl/`

[6] Frank Pfenning. *Lecture Notes on Binary Decision Diagrams*
`https://www.cs.cmu.edu/ fp/courses/15122-f10/lectures/19-bdds.pdf`

[7] Luc De Raedt, Angelika Kimmig and Hannu Toivonen. *ProbLog: A probabilistic Prolog and its Application in Link Discovery*. (Engleză)
Machine Learning, 100:1, pp. 5 - 47, Springer New York LLC, 2015.