Programare Logică Probabilistă

X

October 2018

Cuprins

1 Introducere **3**

2 Problog **4**

2.1 Aspecte generale 4

2.2 Utilizări din linia de comandă 5

1 Introducere

2 Problog

2.1 Aspecte generale

Problog 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*. Astfel, 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.

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

Cu toate acestea, diferența majoră ce deosebește Problog de Prolog se datorează probabilităților ce le putem atribui faptelor. 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 regula "casaDaramata". Astfel, dacă ulterior am vrea să răspundem la întrebarea `query(casaDaramata).` am primi răspunsul `casa Daramata: 0.568` . Răspunsul reprezintă probabilitatea ca intrebarea 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 in 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.

2.2 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 regula 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 ce se află în 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 dispozitie următoarele reprezentări ale unor baze de cunoaștere propozitionale: 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 numarul 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, dupa un 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 regulii 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 întrebarilor 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 raspunsului mai mare, pe când un numar ridicat de exemple pe baza caruiă 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 lângâ faptele anotate cu o anumită probabilitate, putem avea și niste fapte a căror valoare de adevăr o știm cu siguranta, iar acestea sunt reprezentate în program sub forma `evidence(fapt1, true)` sau `evidence(fapt1, false)`. În momentul în care apelam 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 carui 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 underscore.

În final, o ultimă funcționalitate pe care o putem folosi din linia de comandă este `explain`. Acest cuvant cheie deservește cu scopul de a oferi mai multe detalii despre modul în care se calculeaza 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 si 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.