Prolog CheatSheet

Laborator 9

Fapte și Structuri

```
papagal(coco).
iubeste(mihai, maria).
iubeste(mihai, ana).
deplaseaza(scaun, camera1, camera2).
are(ion,carte(aventuri,2002)).
merge(ion,facultate(upb)).
```

Faptele sunt **predicate de ordinul întâi** de aritate n, considerate adevărate.
Structurile sunt înlănțuiri de fapte.
Obținerea primei soluții este de obicei numită satisfacerea scopului iar obținerea altor soluții, resatisfacerea scopului.
Dacă în satisfacerea scopului s-au găsit alternative încă neexplorate pentru satisfacerea unor predicate, se poate cere resatisfacerea scopului tastând **;**. Dacă utilizatorul nu este interesat de resatisfacere poate apăsa tasta **.** sau **Enter**.

```
?- iubeste(mihai, X).
X = maria;  % mai există soluții, utilizatorul poate tasta ;
X = ana. % nu mai există alte soluții, Prolog nu va mai cere
    input de la utilizator.
```

Variabile

```
?- papagal(coco).
true.
?- papagal(CineEste).
CineEste = coco.
?- deplaseaza(_, DeUnde, Unde).
DeUnde = camera1, Unde = camera2
```

Numele argumentelor variabile începe cu literă mare iar numele constantelor simbolice începe cu literă mică.
O variabilă poate fi instanțiată (**legată**) dacă există un obiect asociat acestei variabile, sau neinstanțiată (**liberă**) dacă nu se știe încă ce obiect va desemna variabila.
Wildcard '_' poate fi orice.

Reguli

```
frumoasa(ana).           %1
bun(vlad).               %2
cunoaste(vlad, maria).   %3
cunoaste(vlad, ana).     %4
iubeste(mihai, maria).   %5
iubeste(X, Y):- bun(X),  %6
                cunoaste(X, Y),
                frumoasa(Y).
```

Regula 'iubește(X,Y)' se poate traduce: "X iubește pe Y dacă X e bun și dacă X cunoaște pe Y și dacă Y este frumoasă".
O **regulă** Prolog exprimă un fapt care depinde de alte **fapte**.
Faptele se pot înlănțui folosind virgula(, - **și** logic) și punct și virgula(; - **sau** logic)

Evaluare în Prolog

Prolog este optimist: atunci când Prolog primește o interogare, Prolog încearcă să demonstreze că interogarea este adevărată.
Prolog este perseverent: dacă interogarea conține variabile sau wildcards, încercă să găsească valori pentru elementele neinstanțiate (în domenii care pot fi desprinse din program) în așa fel încât să poată demonstra că interogarea este adevărată.
Prolog face backtracking: în timp ce încearcă demonstrarea unui scop, anumite predicate pot avea mai multe variante de demonstrare (e.g. mai multe reguli, sau folosesc operatori sau). Astfel se creează alternative, care vor fi explorate de Prolog prin backtracking pentru satisfacerea / resatisfacerea scopului.

Operatori

```
Aritmetici: + - * /
Relationali: = \= < > =< >= =:= == =\=
Logici: \+ sau not

?- 1 + 2 =:= 2 + 1.
true.
?- 1 + 2 = 2 + 1.
false. % nu au aceeași formă (nu se evaluează)
?- 2 + 1 = 2 + 1.
true. % au aceeași formă (nu se evaluează)
?- X = 2 + 1.
X = 2+1.
?- X is 2 + 1.
X = 3.
?- X =:= 2 + 1.
ERROR: =:=/2: Arguments are not sufficiently instantiated
```

- \+ echivalent cu **not**, și întoarce true dacă scopul care urmează după nu poate fi satisfăcut în niciun fel (nu se poate demonstra că scopul este adevărat).
- = unifică partea din stânga cu partea din dreapta (dacă este posibil), realizând toate legările necesare. De exemplu: X-Y:Z = 5-[a, b, c]:y unifică astfel X = 5, Y = [a, b, c], Z = y (observați că nu realizează nicio evaluare).
- \= este opusul lui =, și este adevărat dacă partea din stânga nu unifică cu partea din dreapta. Ex: A \= B este echivalent cu (\+ (A = B))
- == verifică dacă partea din stânga și partea din dreapta sunt legate la aceeași valoare.
- =:= evaluează **numeric** expresiile din stânga și dreapta și verifică dacă rezultatele au aceeași valoare numerică. Ambele părți trebuie să fie complet instanțiate.
- =\= evaluează **numeric** expresiile din stânga și dreapta și returnează true dacă valorile sunt diferite. Ambele părți trebuie să fie complet instanțiate.
- **is** evaluează numeric partea dreaptă (care trebuie să fie complet instanțiată, și:
  - dacă partea stângă este instanțiată și este o valoare, verifică egalitatea valorilor.
  - dacă partea stângă este o variabilă, leagă variabila la valoarea din dreapta.
  - altfel, false.

Liste

```
[]                  - lista vida
[a,b,c]             - elementele a, b, c
[Prim|Rest]         - element concatenat la rest
[X1,X2,...,XN|Rest] - n elemente concatenate la restul listei
```

Pattern matching

```
Haskell:
sum []     = 0              %1
sum (x:xs) = x + sum xs     %2

Prolog: (pentru predicatul sum(+L, -Sum)
sum([], 0).       % dacă primul argument este lista vidă, al
    doilea argument trebuie să fie / este legat la 0
?- sum([H|T], S) :- sum(T, ST), S is ST + H. % dacă primul
    argument este o listă construită prin adăugarea unui element
    H la începutul unei liste T, și dacă suma listei T este ST, S
    este ST plus H.
```

În **Haskell** se realizează pattern matching intr-o singură direcție: odata legate variabilele, se produce acțiunea descrisă în dreapta.
În **Prolog** pattern matching se efectuează în ambele direcții: folosirea variabilelor pentru a executa regulile și produce un raspuns **true** sau **false**, sau se pot căuta atât variabile care satisfac regulile scrise.

Errors/Warnings

```
my_length1([],0).
my_length1([H|L],N) :- my_length1(L,N1), N1 is N - 1.

?- my_length([a, b, c])
Error: Arguments are not sufficiently instantiated
```

N nu are o valoare instanțiată pentru a putea scădea 1 și a da valoarea lui N1. N1 va primi valoarea 0 din apelul my_length2([], 0), iar N trebuie calculat în funcție de N1 existent.

```
my_length2([],0).
my_length2([H|L],N) :- my_length2(L,N1), N is N1 + 1.

Warning: Singleton Variables: [H]
```

O variabilă singleton este o variabilă care apare o singură dată într-o regulă. Dacă variabila nu apare de mai multe ori în regulă, înseamnă că numele respectiv nu este folosit pentru a transmite o legare dintr-o parte în alta, deci este inutil.

```
Correct:
my_length3([],0).
my_length3([_|L],N) :- my_length3(L,N1), N is N1 + 1.
```

Documentarea predicatelor și a argumentelor

```
predicat/nrArgumente
predicat(+Arg1, -Arg2, ?Arg3, ..., +ArgN)
```

Pentru a diferenția intrările **(+)** de ieșiri **(-)**, se prefixează argumentele cu indicatori. Acele argumente care pot fi fie intrări, fie ieșiri se prefixează cu **(?)**
Unele predicate nu au parametrii
Ex: Predicatul **fail/0** care se va evalua mereu la **false**.