Deklaratív programozás, 2. gyakorlat
Prolog programozás
2011. 09. 30.

---

1. Témakör: Prolog végrehajtás

Definiáljuk egy személy felmenőjét a következő két állítással:

X felmenője Y, ha Y X egyik szülőjének felmenője.
X felmenője X.

Tekintse a fenti definíciót megvalósító és a "szülője" kapcsolatot is leíró alábbi Prolog programot.

```
% f(X, Y): X felmenője Y.
f(X, Y) :- sz(X, S), f(S, Y).     % (f1)
f(X, X).                          % (f2)

% sz(X, Y): X szülője Y.
sz(a, b).                         % (sz1)
sz(a, c).                         % (sz2)
sz(c, d).                         % (sz3)
```

1a. Rajzolja fel a
```
| ?- f(a, V).                                          (1)
```
kérdéshez tartozó keresési fát! A keresési fa alapján írja le, hogy a Prolog rendszer erre a kérdésre milyen válaszokat ad, és milyen sorrendben adja ezeket!

1b. (szorgalmi, otthoni feladat) Rajzolja fel a | ?- f(U, V). kérdéshez tartozó keresési fát; ennek alapján írja le, hogy a Prolog rendszer erre a kérdésre milyen válaszokat ad, és milyen sorrendben adja ezeket!

1c. Adja meg, hogy az (1) kérdésre milyen sorrendben kapjuk meg a válaszokat, ha a fenti programban az (f1) és (f2) klózok sorrendjét megcseréljük!

1d. (szorgalmi, otthoni feladat) Mi történik az (1) kérdés futtatásakor, ha az (f1) és (f2) klózok ebben a sorrendben követik egymást, de az (f1) klóz törzsében levő két hívás sorrendjét megcseréljük?

1e. Rajzolja fel az f predikátum eredeti alakjának eljárásdobozát! Megjegyzés: ehhez a második klózt olyan alakra kell hoznia, hogy az elsővel azonos feje legyen:

```
f(X, Y) :- sz(X, S), f(S, Y).
f(X, Y) :- Y = X.
```

1f. (szorgalmi, otthoni feladat) Az 1e. feladatra adott megoldása alapján írja le az (1) cél futásához tartozó eljárásdobozos nyomkövetést, azaz azt, hogy az (1) cél futása során milyen kapukon milyen eljáráshívásokkal halad át a vezérlés. Az f eljárás rekurzív hívásait nem kell részleteznie: ezeket lépje át, mintha a nyomkövetésben az ‘s’ (skip) parancsot adná ki. Az 1a. feladatra adott megoldását felhasználhatja arra, hogy megállapítsa, hogy az átlépett eljáráshívás hogyan helyettesíti be a kimenő paramétereit. Az indexelés hatásával, a determinisztikus lefutással ne foglalkozzon!

Az alábbiakban megadjuk a nyomkövetés elejét, ezt az adott formában folytassa!

```
  Call: f(a,V)              (c1)
    Call: sz(a,S)           (c2)
    Exit: sz(a,b)
    Call: f(b,V)  skip      (c3)
    Exit: f(b,b)
  Exit: f(a,b)
V = b ? ;
  Redo: f(a,b)
    Redo: f(b,b)  skip
    Fail: f(b,V)
    ...
```

1g. (szorgalmi, otthoni feladat) Tegye teljessé az 1f. feladatra adott megoldását, úgy hogy a korábban "skip" paranccsal átlépett részeket is kifejti. A nyomkövetésben Jelölje meg a Call kapukat, azaz folytassa a megkezdett (c1), (c2), ... címkézést. Az 1a. feladat megoldásaként megrajzolt keresési fában keresse meg az egyes Call kapukhoz tartozó csomópontokat, és lássa el a megfelelő címkével. A keresési fában lehet-e valamit megfeleltetni az Exit, Fail, ill. Redo kapukon való áthaladásnak?

---

2. Témakör: szintaktikus édesítőszerek, egyesítés

Írja fel az alábbi egyenlőségek bal- és jobboldalának alapstruktúra alakját, vagy rajzolja fel a fastruktúrájukat! Adja meg, milyen változó-behelyettesítéseket eredményeznek ezek az egyesítések!

2a. `[a*b-X,c+X] = [Y-3|Z].`

2b. `g(V*W, [1*2+3|Z]) = g(K, [K+L,L]).`

2c. (szorgalmi, otthoni feladat)
`s([a]-X/2, [X|Z]) = s(Z-Y, [f,C]).`

2d. (szorgalmi, otthoni feladat)
`[A+B, C*2|T] = .(x/C+2, [y*B, B-C]).`

---

3. Témakör: programok írása

A programozási feladatok megoldásaként olyan Prolog eljárást kell írnia, amely megfelel az adott fejkommentnek. Ha külön nem kérjük, akkor nem szükséges, hogy jobbrekurzív (iteratív) eljárásokat írjon, de természetesen a jobbrekurzív változatot ilyenkor is elfogadjuk.

Bármely feladat megoldásához felhasználhat korábbi sorszámú feladatokban definiált eljárásokat. Ha ilyeneken kívül is szükséges segédeljárás definiálása, azt külön jelezzük. Természetesen más esetben is használhat segédeljárást. Minden segédeljáráshoz írjon fejkommentet!

Hibakezeléssel nem kell foglalkoznia: a megírt eljárásoknak csak a fejkomment által megadott argumentumértékek esetén kell az előírt módon viselkedniük.

Az alábbi példasorban a (bináris) fa adatstruktúra alatt egy olyan Prolog kifejezést értünk, amely lehet
- levél, azaz egy ‘leaf’ nevű struktúra, amelynek egyetlen argumentuma egy egész szám, pl. leaf(1), leaf(2); vagy
- csomópont, azaz egy node nevű kétargumentumú struktúra, amelynek mindkét argumentuma ‘fa’, pl. node(leaf(1),leaf(2)).

A fejkommentekben használt ún. input/output módjelölések magyarázata:

*: az argumentum tömör bemenő, azaz nem lehet változó, és nem is tartalmazhat változót;
+: az argumentum bemenő, azaz nem lehet változó, de tartalmazhat változót (természetesen csak akkor, ha az argumentum egy struktúra);
-: az argumentum kimenő, azaz változó;
?: az argumentum lehet kimenő és bemenő is.

3. Hatványozás

```
% hatv(+A, +E, -H):  H = A ^ E, ahol A egész szám, E >= 0 egész szám.

| ?- hatv(3, 5, X).
X = 243 ? ; no
```

4. Fa csomópontjainak megszámolása

Egy fa csomópontjainak száma a benne előforduló node/2 struktúrák száma.

```
% fa_pontszama(*Fa, −N): A Fa bináris fa csomópontjainak száma N.

| ?− fa_pontszama(node(leaf(1),node(leaf(2),leaf(3))), N).
N = 2 ? ; no
| ?− fa_pontszama(node(leaf(1),node(leaf(2),node(leaf(4),leaf(3)))), N).
N = 3 ? ; no
```

5. Fa minden levélértékének növelése

```
% fa_noveltje(*Fa0, ?Fa): Fa úgy áll elő a Fa0 bináris fából, hogy az
% utóbbi minden egyes levelében levő értéket 1−gyel megnöveljük.

| ?− fa_noveltje(node(leaf(1),node(leaf(2),leaf(3))), Fa).
Fa = node(leaf(2),node(leaf(3),leaf(4))) ? ; no
```

6. Lista hosszának meghatározása

Egy lista hosszának az elemei számát nevezzük.

```
% lista_hossza(*Lista, −Hossz): A Lista egészlista hossza Hossz.

| ?− lista_hossza([1,3,5], H).
H = 3 ? ; no
```

6*. (szorgalmi, otthoni feladat) Lista hosszának meghatározása −− jobbrekurzív változat

```
% lista_hossza2(*Lista, −Hossz): A Lista egészlista hossza Hossz.
% Jobbrekurzív változat
Segédeljárás szükséges.
```

7. Egészlista minden elemének növelése

```
% lista_noveltje(*L0, ?L): Az L egészlista úgy áll elő az L0
% egészlistából, hogy az utóbbi minden egyes elemét 1−gyel megnöveljük.

| ?− lista_noveltje([1,5,2], L).
L = [2,6,3] ? ; no
```

8. Egy lista utolsó elemének meghatározása

```
% lista_utolso_eleme(*L, ?Ertek): A Lista egészlista utolsó eleme Ertek.

| ?− lista_utolso_eleme([5,1,2,8,7], E).
E = 7 ? ; no
```

9. Egy fa leveleiben található értékek felsorolása

```
% fa_levelerteke(*Fa, −Ertek): A Fa bináris fa egy levelében található
% érték az Ertek.
```

Az eljárás nemdeterminisztikus módon sorolja fel az összes levélértéket. A felsorolás sorrendjére nem teszünk megkötést.

```
| ?− fa_levelerteke(node(leaf(1),node(leaf(2),leaf(3))), E).
E = 1 ? ; E = 2 ? ; E = 3 ? ; no
```

10. Egy fa részfáinak a felsorolása

Egy fa (nem feltétlenül valódi) részfájának nevezzük saját magát, valamint − ha a fa egy csomópont − akkor a bal és jobboldali ág részfáit.

```
% fa_reszfaja(*Fa, −Resz): Resz a Fa bináris fa részfája.
```

A fenti eljárás nemdeterminisztikus, azaz többféleképpen sikerül: a Resz változóban fel kell sorolnia a Fa összes részfáját. A felsorolás sorrendjére nem teszünk megkötést.

```
| ?− fa_reszfaja(node(leaf(1),node(leaf(2),leaf(3))), Fa).
Fa = node(leaf(1),node(leaf(2),leaf(3))) ? ;
Fa = leaf(1) ? ;
Fa = node(leaf(2),leaf(3)) ? ;
Fa = leaf(2) ? ;
Fa = leaf(3) ? ; no
```

Gondolja meg, hogy a predikátum klózai sorrendjének változtatásakor hogyan változik a felsorolás sorrendje!

A fa_reszfaja eljárás felhasználásával írja meg a 9. feladat megoldását, fa_levelerteke2 néven!

11. Egy lista prefixumainak a felsorolása

Egy L n−elemű lista prefixumának nevezzünk egy listát, ha az az L első k elemét tartalmazza (az L−beli sorrend megtartásával), ahol 0 =< k =< n.

```
% lista_prefixuma(*L0, −L): L az L0 egészlista prefixuma.
```

A fenti eljárás nemdeterminisztikus, azaz többféleképpen sikerül: az L változóban fel kell sorolnia a L0 összes prefixumát. A felsorolás sorrendjére nem teszünk megkötést.

```
| ?− lista_prefixuma([1,4,2], Sz).
Sz = [1,4,2] ? ;
Sz = [1,4] ? ;
Sz = [1] ? ;
Sz = [] ? ; no
```

Gondolja meg, hogy a predikátum klózai sorrendjének változtatásakor hogyan változik a felsorolás sorrendje!

Szorgalmi, otthoni programozási feladatok
==========

+1. Fa mélységének meghatározása

Egy fa mélységén az egymásba skatulyázott node struktúrák maximális számát értjük.

```
% fa_melysege(*Fa, ?M): A Fa bináris fa mélysége M.

| ?- fa_melysege(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), N).
N = 2 ? ; no
| ?- fa_melysege(node(leaf(1),node(leaf(2),node(leaf(4),leaf(3)))), N).
N = 3 ? ; no
```

Tipp: az aritmetikai kifejezésekben megengedett a max függvény használata, pl | ?- X is max(2,3)+1. ---> X = 4 ? ; no

+2. Egy fa legbaloldalibb levélértékének meghatározása

```
% fa_balerteke(*Fa, ?Ertek): A Fa bináris fa legbaloldalibb levelében
% az Ertek egész érték szerepel.

| ?- fa_balerteke(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), E).
E = 1 ? ; no
```

+3. Egy fa legjobboldalibb levélértékének meghatározása

```
% fa_jobberteke(*Fa, ?Ertek): A Fa bináris fa legjobboldalibb levelében
% az Ertek egész érték szerepel.

| ?- fa_jobberteke(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), E).
E = 3 ? ; no
```

+4. Egy fa rendezettségének eldöntése

Egy fát rendezettnek mondunk, ha a levelek értékei, balról jobbra haladva szigorúan monoton növő sorozatot alkotnak.

```
% fa_rendezett(*Fa): A Fa bináris fa rendezett.

| ?- fa_rendezett(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3)))).
no
| ?- fa_rendezett(node(node(leaf(1),leaf(3)),
                       node(leaf(5),node(leaf(6),leaf(9))))).
yes
```

A megoldásban célszerű az előző két feladat eljárásait segédeljárásként felhasználni, így nem szükséges további segédeljárást definiálni.

Keressen hatékonyabb megoldást is, amelyben a fastruktúrát csak egyszer járja be! Ebben feltételezheti, hogy a fa csak pozitív számokat tartalmaz (fa_rendezett2). Ehhez a megoldáshoz szükség lehet egy megfelelő segédeljárásra.

+5. Fa tükörképének képzése

```
% fa_tukorkepe(*Fa0, ?Fa): Fa a Fa0 bináris fa tükörképe.

| ?- fa_tukorkepe(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), Fa).
Fa = node(node(leaf(3),leaf(2)),node(leaf(4),leaf(1))) ? ; no
| ?- fa_tukorkepe(node(node(leaf(1),leaf(4)),node(leaf(4),leaf(1))), Fa).
Fa = node(node(leaf(1),leaf(4)),node(leaf(4),leaf(1))) ? ; no
```

+6. Fa tükörszimmetrikus voltának ellenőrzése

```
% fa_tukros(*Fa): A Fa bináris fa tükörszimmetrikus.

| ?- fa_tukros(node(node(leaf(1),leaf(4)),node(leaf(4),leaf(1)))).
yes
| ?- fa_tukros(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3)))).
no
```

+7. Lista adott hosszú prefixuma

Egy L n-elemű lista prefixumának nevezünk egy listát, ha az az L első k elemét tartalmazza (az L-beli sorrend megtartásával), ahol 0 =< k =< n.

```
% prefix_length(+Whole, ?Prefix, +Length): A Whole lista
% prefixuma a Prefix lista, amelynek hossza Length.
| ?- prefix_length([a,b,c,d,e], Prefix, 3).
Prefix = [a,b,c] ? ;
no
```

+8. Lista adott helyen kezdődő szuffixuma

Egy L lista szuffixumának nevezünk egy listát, ha az az L utolsó valahány elemét tartalmazza, az L-beli sorrend megtartásával.

```
% suffix_before(+Whole, ?Suffix, +Before): A Whole zárt végű lista
% azon szuffixuma a Suffix lista, amelyet megelőző elemek száma
% Before.
| ?- suffix_before([a,b,c,d,e], Suffix, 3).
Suffix = [d,e] ? ;
no
```

+9. Részlista képzése

```
% sublist(+Whole, ?Part, +Before, +Length): A Whole zárt végű
% lista azon (folytonos) részlistája Part, amely előtt Before
% számú elem áll és amelynek hossza Length.
| ?- sublist([a,b,c,d,e], Part, 1, 3).
Part = [b,c,d] ? ;
no
```

+10. Legnagyobb közös osztó -- euklideszi algoritmus

Írjon egy az euklideszi algoritmust megvalósító Prolog eljárást!

```
% lnko(+A, +B, -LNKO): LNKO az A és B nem-negatív számok legnagyobb
% közös osztója (de A és B nem lehet egyaránt 0).

% Az A egész szám B-vel való osztásának M maradékát az 'M is A mod B'
% hívással állíthatja elő (A >=0, B > 0).

| ?- lnko(24, 33, X).
X = 3 ? ; no
```