Programare Logică

Structuri de date. Lise. Recursivitate.

- Liste
- Recursivitate

Liste

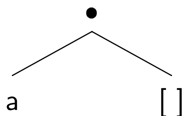
- Listele sunt o structură de date folosită adesea în calculul simbolic.
- Listele conțin elemente care sunt ordonate.
- Elementele listelor sunt termeni (orice tip, inclusiv alte liste).
- Listele sunt singura structură de date în LISP
- Sunt *o* structură de date în Prolog.
- Listele pot reprezenta practic *orice* structură.

Liste (domeniu inductiv)

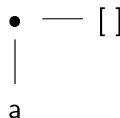
- “Cazul de bază”: [ ] – lista goală.
- “Cazul general” : .(h, t) – lista nevidă, unde:
  - h - capul, care poate fi orice termen,
  - t - coada, trebuie să fie o listă.

Reprezentarea listelor

- .(a, [ ]) e reprezentată ca



or

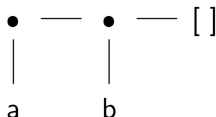


"tree representation"

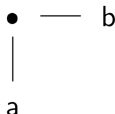
"vine representation"

Reprezentarea listelor

- .(a, .(b, [ ])) este

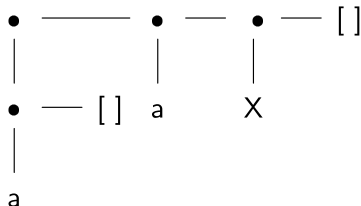


- .(a, b) *nu* este o listă, dar e o structură legală Prolog, reprezentată ca



Reprezentarea listelor

- .(.(a, []), .(a, .(X, [ ]))) e reprezentată ca



Syntactic sugar pentru liste

- Pentru a simplifica notația, putem folosi "," pentru a separa elementele.
- Listele introduse anterior sunt:

```
[a],
[a, b],
[[a], a, X].
```

Manipularea listelor

- Listele sunt împărțite în cap și coadă.
- Prolog oferă o construcție pentru a profita de acest lucru: [H | T].
- Considerăm următorul exemplu:

```
p([1, 2, 3]).
p([o, pisica, [pe, saltea]]).

?-p([H | T]).
     H = 1,
     T = [2, 3];
     H = o
     T = [pisica, [pe, saltea]];
     no
```

- Atenție [a | b] nu este o listă, dar e o expresie Prolog validă, care corespunde la .(a, b)

Unificarea listelor: exemple

```
[X, Y, Z] = [ioan, vrea, peste]
    X = ioan
    Y = vrea
    Z = peste

[pisica] = [X | Y]
    X = pisica
    Y = [ ]

[X, Y | Z] = [maria, vrea, vin]
    X = maria
    Y = vrea
    Z = [vin]
```

Unificarea listelor: exemple

```
[[un, Y] | Z] = [[X, iepure], [e, aici]]
    X = un
    Y = iepure
    Z = [[e, aici]]

[mare | T] = [mare, albastra]
    T = [albastra]


[iarna, grea] = [grea, X]
    false

[alb|Q] = [P|negru]
    P = alb
    Q = negru
```

Șiruri de caractere

- În Prolog, șirurile de caractere sunt scrise între ghilimele.
- Exemplu: "un sir".
- În reprezentarea internă, un șir este o listă care conține codul ASCII corespunzător fiecărui caracter din șir.
- ```
  ?- X = "un sir".
     X=[117, 110, 32, 115, 105, 114].
  ```
  ```

Sumar

- anatomia unei liste în Prolog `.(h, t)`
- reprezentarea grafică a listelor: “tree representation”, “vine representation”,
- syntactic sugar pentru liste `[...]`,
- manipularea listelor: notația head-tail [H|T],
- șirurile de caractere ca liste,
- unificarea listelor.

Inducție/Recursivitate

Un domeniu inductiv:

- Un domeniu compus din obiecte construite într-un mod "ușor de gestionat" și anume:
- sunt câteva obiecte atomice "cele mai simple", care nu se mai pot descompune,
- sunt obiecte "complexe" care pot fi descompuse într-un număr **finit** de obiecte mai simple,
- iar acest proces de descompunere se poate repeta de un număr finit de ori înainte de a ajunge la "cele mai simple" obiecte.
- În asemenea domenii putem utiliza **inducția** ca regulă de inferență.

Inducție/Recursivitate

Recursivitatea e duală inducției:

- recursivitatea descrie calcule într-un domeniu inductiv,
- procedurile recursive (funcții, predicate) se autoapelează,
- DAR apelul recursiv trebuie să se facă pe un obiect "mai simplu".
- Ca rezultat, o procedură recursivă va trebui să descrie comportamentul pentru:

(a) "Cel mai simplu" obiect, și/sau obiecte/situații pentru care calculele se opresc, cazul de bază, și

(b) cazul general, care descrie apelul recursiv.

Exemplu: liste ca domeniu inductiv

- cel mai simplu obiect: lista vidă [ ].
- orice altă listă e formată din cap și coadă (coada trebuie să fie listă): [H|T].

Exemplu: membru

- Implementați în Prolog predicatul membru/2, astfel încât membru(X, Y) e adevărat când X se găsește în lista Y.

Exemplu: membru

- Implementați în Prolog predicatul membru/2, astfel încât membru(X, Y) e adevărat când X se găsește în lista Y.

```
% Cazul de baza.
membru(X, [X|_]).
% Cazul general (apel recursiv).
membru(X, [_|Y]):-
            membru(X, Y).
```

Exemplu: membru

- Implementați în Prolog predicatul membru/2, astfel încât membru(X, Y) e adevărat când X se găsește în lista Y.

```
% Cazul de baza.
membru(X, [X|_]).
% Cazul general (apel recursiv).
membru(X, [_|Y]):-
                membru(X, Y).
```

- Cazul de bază, în acest exemplu, este condiția pentru care calculele se opresc (nu este pentru ”cea mai simplă” listă, adică pentru [ ]).

Exemplu: membru

- Implementați în Prolog predicatul membru/2, astfel încât membru(X, Y) e adevărat când X se găsește în lista Y.

```
% Cazul de baza.
membru(X, [X|_]).
% Cazul general (apel recursiv).
membru(X, [_|Y]):-
                membru(X, Y).
```

- Cazul de bază, în acest exemplu, este condiția pentru care calculele se opresc (nu este pentru "cea mai simplă" listă, adică pentru [ ]).
- Pentru [ ] nu am specificat comportamentul predicatului (unde eșuează).

Exemplu: membru

- Implementați în Prolog predicatul membru/2, astfel încât membru(X, Y) e adevărat când X se găsește în lista Y.

```
% Cazul de baza.
membru(X, [X|_]).
% Cazul general (apel recursiv).
membru(X, [_|Y]):-
                membru(X, Y).
```

- Cazul de bază, în acest exemplu, este condiția pentru care calculele se opresc (nu este pentru "cea mai simplă" listă, adică pentru [ ]).
- Pentru [ ] nu am specificat comportamentul predicatului (unde eșuează).
- Apelul recursiv se face pe o listă mai mică (al doilea argument). Elementele în apelul recursiv devin din ce în ce mai mici în așa fel încât calculele ajung la succes sau ajung la lista vidă sau eșuează.

Când folosim recursivitatea?

- A se evita definiții circulare:

```
parinte(X, Y):- copil(Y, X).
copil(X, Y):- parinte(Y, X).
```

- Atenție cu recursivitatea la stânga:

```
persoana(X):-persoana(Y), mama(X, Y).
persoana(adam).
```

În acest caz,

```
?-persoana(X).
```

va rula la infinit. Prolog încearcă să satisfacă regula, iar acest lucru duce la “Out of local stack”.

- Ordinea faptelor și a regulilor în baza de cunoștințe:

```
e_lista([A|B]) :- e_lista(B).
e_lista([]).
```

Următoare întrebare duce la ciclu infinit:

```
?- e_lista(X)
```

- Ordinea în care sunt scrise regulile și faptele contează! În general, **scriem faptele înaintea regulilor**.

Exerciții

- Definiți predicate în Prolog pentru:

Exerciții

- Definiți predicate în Prolog pentru:
  1. Lungimea unei liste

Exerciții

- Definiți predicate în Prolog pentru:
    1. Lungimea unei liste
    2. Suma elementelor unei liste

Exerciții

- Definiți predicate în Prolog pentru:
  1. Lungimea unei liste
  2. Suma elementelor unei liste
  3. Inversa unei liste

Exerciții

- Definiți predicate în Prolog pentru:
  1. Lungimea unei liste
  2. Suma elementelor unei liste
  3. Inversa unei liste
  4. Lista elementelor de pe pozițiile pare

Exerciții

- Definiți predicate în Prolog pentru:
  1. Lungimea unei liste
  2. Suma elementelor unei liste
  3. Inversa unei liste
  4. Lista elementelor de pe pozițiile pare
  5. Concatenarea a două liste.