

רשימות הפרש

תוספת לפרק 8


הבעיה (1):

- שרשור רשימות (conc) :

`conc([],L,L).`

`conc([X|Xs],L1,[X|L2]):-`

`conc(Xs,L1,L2).`



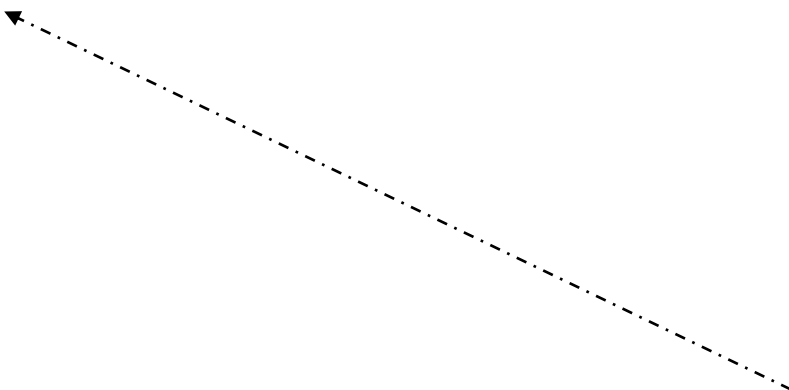
פעולה שימושית מאד ופשוטה
בסיבוכיות יקרה מדי !

רשימות הפרש

הבעיה (2):

- גישה לסוף הרשימה –
(למשל האיבר האחרון ברשימה):

```
last([X],X):- !.  
last([_|Xs],X):-  
    last(Xs,X).
```



Cut אדום להזכירכם

סיבוכיות :

- סיבוכיות הזמן של שתי הפעולות האחרונות (כמו של רוב הפעולות שכרוכות בגישה לסוף הרשימה) היא :
 $O(n)$
- כש n היא אורך הרשימה .
- זאת סיבוכיות זמן גבוהה מדי לשימוש בפעולה בסיסית כל כך מבחינתנו .
- ננסה למצוא דרך לייעל תהליכים מסוג זה :

פתרון חלקי :

- כזכור, מנקודת המבט הרקורסיבית שלנו אל רשימות, הן מסתיימות ברשימה ריקה .

- וכך לדוגמא :

$$[1,2,3] = [1 | [2,3]] = [1,2, | [3]] = [1,2,3 | []]$$

- מה אם היתה בידינו רשימה שמסתיימת ב - "משתנה" (ערך לא מאתחל) :

$$[1,2,3|\text{Tail}]$$

אז :

? - $L = [1,2,3|Tail]$, $Tail = [8,9]$.



• ונקבל :

$L=[1,2,3,8,9]$.

- פעולת ההתאמה שבוצעה כאן היא בסיבוכיות קבועה ולכן יש כאן שיפור של מדד היעילות בסדר גודל שלם .
- מצאנו דרך לייעל את העבודה על רשימות .

רשימות ההפרש :

- משיקולי גמישות לשינויים (יצירה ופירוק של רשימות כאלו) נשתמש ברשימות הפרש :
- רשימת הפרש היא רשימה שזנבה הוא משתנה חופשי ולאחריו הסימן '-' ומופע נוסף של המשתנה החופשי .
- וכך רשימה מקבילה ל [1,2,3] תהיה למשל :

[1,2,3|Tail] - Tail

ואז ...

- נוכל לבצע פעולות שונות שקשורות בסוף רשימת הפרש בסיבוכיות זמן קבועה , כמו למשל שרשור ופעולת גישה לסוף הרשימה (נדגים בהמשך).
- חשוב לזכור כי למרות ששם מבנה הנתונים נגזר מאותו סימן '-' בין שני מרכיביו זהו לא מבנה נתונים שמוכר על ידי המהדר ולמעשה נוכל להחליף סימן זה בכל סימן מפריד אחר .

שרשור :

- **conc(L1-T1,T1-T2,L1-T2).**

| ?- conc([1,2,3|X]-X,[8,9|Y]-Y,L-Tail).

X = [8,9|Y] ,

Y = Tail = _ ,

L = [1,2,3,8,9|Y]

| ?- conc([1,2,3|X]-X,[8,9|Y]-Y,L-[]).

X = [8,9] ,

Y = [] ,

L = [1,2,3,8,9]

איך זה עובד ?

- $\text{conc}(\text{L1-T1}, \text{T1-T2}, \text{L1-T2})$.


הכול עניין של התאמות , לדוגמא :

$\text{conc}([1,2,3|X] - X , [8,9|Y] - Y, L - \text{Tail})$.

- $\text{conc}(\text{L1}-\text{T1}, \text{T1}-\text{T2}, \text{L1}-\text{T2}).$

התאמה ראשונה :

$\text{conc}([1,2,3|X] - \text{X}, [8,9|Y] - Y, L - \text{Tail}).$



$T1 = X = [8,9|Y]$

- **conc(L1-T1,T1-T2,L1-T2).**

: TAXI

conc([1,2,3|**X**] - **X** , [8,9|Y] - Y, L - Tail).



$L1 = [1,2,3|[8,9|Y]]$

- **conc(L1-T1,T1-T2,L1-T2).**

או בעצם :

conc([1,2,3|X] – X , [8,9|Y] - **Y**, L - Tail).

L1 = [1,2,3,8,9|Y]

T2 = Y

- **conc(L1-T1,T1-T2,L1-T2).**

והתוצאה הסופית

conc([1,2,3|X] – X , [8,9|Y] - Y, **L - Tail**).


$L = L1 = [1,2,3,8,9|Y]$

$Tail = T2 = Y$

- **conc(L1-T1,T1-T2,L1-T2).**

ואם נציב :

conc([1,2,3|X] – X , [8,9|Y] – Y, L – []).



נקבל רשימה רגילה בתוצאה :

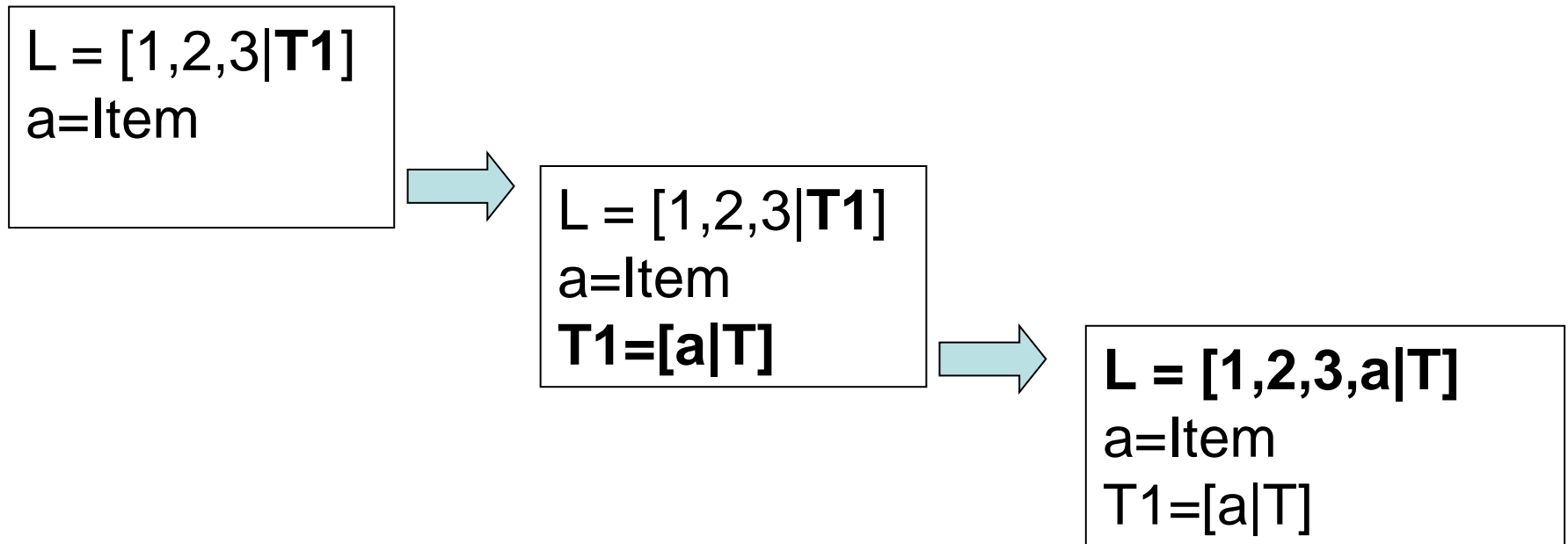
L = L1 = [1,2,3,8,9]

Tail = Y = []

הוספת איבר לסוף הרשימה :

add_to_end(L- [Item|T] ,Item ,L - T).

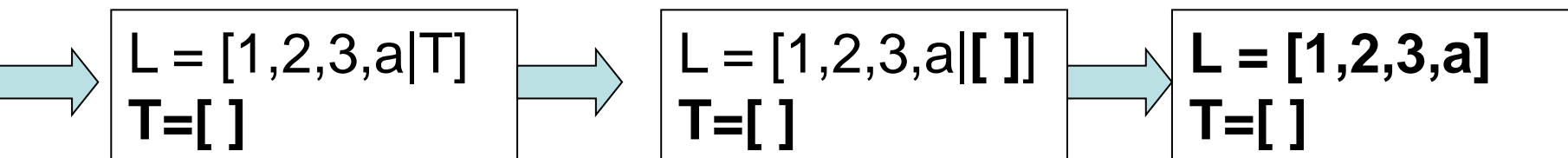
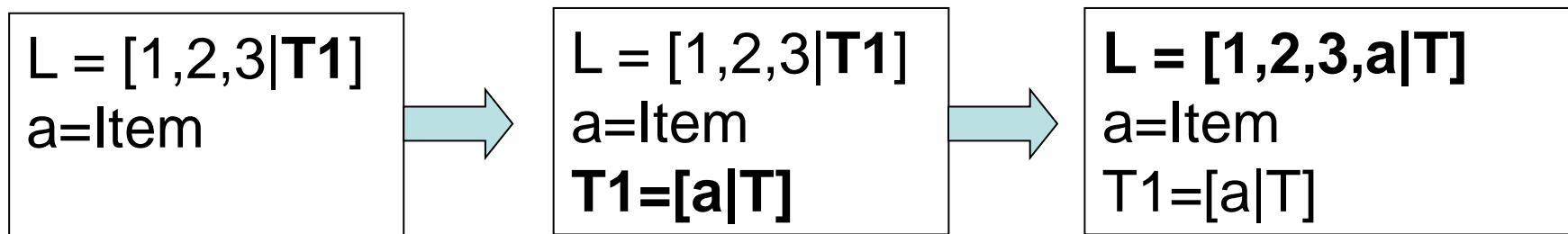
? - add_to_end([1,2,3|T1] - T1,a,Res - T2).



ואם נבצע התאמת רשימה ריקה לזנב :

add_to_end(L- [Item|T] ,Item ,L - T).

| ?- add_to_end([1,2,3|T1]-T1,a,Res-[]).



```
*/  
-? |add2_2_end([1,2,3,4|T1]-T1,sss,eee,Res-  
[]).
```

```
T1 = [sss,eee] ,
```

```
Res = [1,2,3,4,sss,eee]
```

```
/*
```

```
add2_2_end(L1-[Item1,Item2|Tail],Item1,Item2,L1-Tail).
```

זיהוי סוף הרשימה :

- נתבונן בדוגמא הבאה :

length_dl(L- L,0):-!.

**length_dl([X|T] - L,N):-
length_dl(T- L,N1),
N is N1 + 1 .**

- היחס אמור להחזיר את אורך הרשימה .



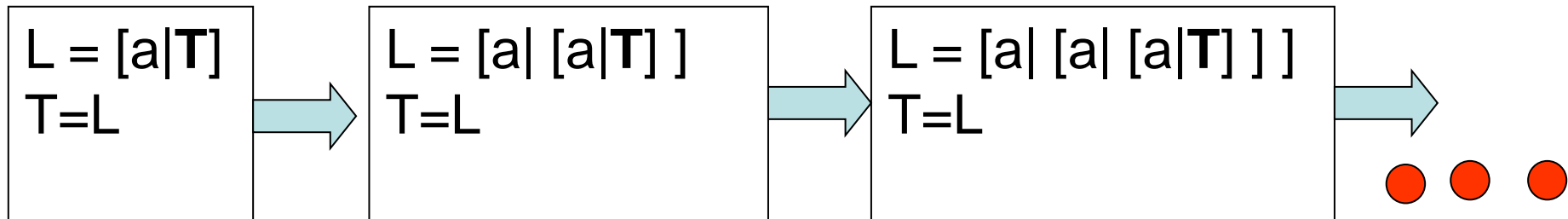
זיהוי סוף הרשימה :

• בפועל מה שיקרה הוא שהתנאי :

$\text{length_dl}(L-L, 0) :- !.$

יגרום ללולאה אין סופית , בדומה להצבת מראה אל מול מראה .

? - $\text{Length_dl}([a|T]-T, N).$



זיהוי סוף הרשימה – הפתרון :

length_dl(T-L,0):-

T == L,!.

length_dl([X|T]-L,N):-

length_dl(T-L,N1),

N is N1 + 1 .

nonmember

non-member(X,L-T):-

 L==T,!.

non-member(X,[Y|L]-T):-

 X\=Y,

 non-member(X,L-T).

היפוך רשימה :

reverse1(A-Z,L-L):-

A == Z , !.

reverse1([X|Xs] - Z,L - T):-

reverse1(Xs - Z, L - [X|T]).

| ?- reverse1([1,2,3,4|Z]-Z,Res-L).

Z = _ ,

Res = [4,3,2,1|L] ,

L = _

| ?- reverse1([1,2,3,4|Z]-Z,Res-[]).

Z = _ ,

Res = [4,3,2,1]

החזרת כל האיברים הגדולים מ 5

```
find-bigger-then-5(L-T,X-X):-
```

```
    L==T,!.
```

```
find-bigger-then-5([X|Xs]-T,[X|L1]-T1):-
```

```
    X>5,!.
```

```
    find-bigger-then-5( Xs-T, L1 -T1).
```

```
find-bigger-then-5([_|Xs]-T, L1 -T1):-
```

```
    find-bigger-then-5( Xs-T, L1 -T1).
```

/*

|| ?- count5([11,2,322,4|T1]-T1,N).

T1 = _ ,

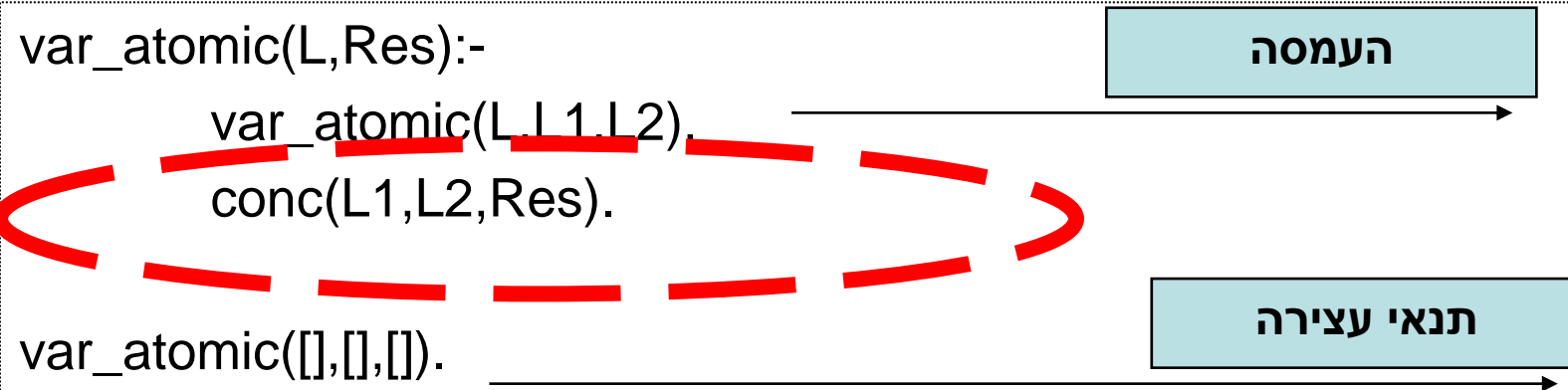
N = 2

*/

- count5(T-L,0):-
- T==L,!.
- count5([X|Xs]-T,N):-
- count5(Xs - T,N1),
- (X>5,! ,N is N1 +1
- ;
- N is N1).

עבודה מעשית ...

פרדיקט המקבל רשימה וממין אותה משתנים לתחילתה ואטומים
או מספרים לסופה :



`var_atomic([X|Xs],[X|Rest1],L2):-`
 `var(X),!,var_atomic(Xs,Rest1,L2).`

`var_atomic([X|Xs],L1,[X|Rest2]):-`
 `atomic(X),var_atomic(Xs,L1,Rest2).`

עבודה מעשית ...

פרדיקט המקבל רשימה וממין אותה משתנים לתחילתה ואטומים
או מספרים לסופה :

`var_atomic(L,Res):-`

`var_atomic(L,L1-T1,L2-T2),
conc(L1-T1,L2-T2,Res-[]).`

העמסה

`var_atomic([],T1-T1,T2-T2).`

תנאי עצירה

`var_atomic([X|Xs],[X|Rest1]-T1,L2-T2):-`

`var(X),!,var_atomic(Xs,Rest1-T1,L2-T2).`

`var_atomic([X|Xs],L1-T1,[X|Rest2]-T2):-`

`atomic(X),var_atomic(Xs,L1-T1,Rest2-T2).`

: וא

| ?- var_atomic([1,2,X,bob,A,roy],Res).

X = _ ,

A = _ ,

Res = [X,A,1,2,bob,roy]

ובעוד דרך – נטמיע את השרשור

`var_atomic(L,Res):-`

`var_atomic(L,Res-Tail,Tail-[]).`

איך תתבצע ההשמה ?

`var_atomic([],T1-T1,T2-T2).`

`var_atomic([X|Xs],[X|Rest1]-T1,L2-T2):-`

`var(X),!,var_atomic(Xs,Rest1-T1,L2-T2).`

`var_atomic([X|Xs],L1-T1,[X|Rest2]-T2):-`

`atomic(X),var_atomic(Xs,L1-T1,Rest2-T2).`

דוגמאות ריצה :

```
| ?-var_atomic([1,2,3,4],Res) .  
Res = [1,2,3,4]
```

```
| ?- var_atomic([roy,1,2,X,Roy,3,4],Res) .  
X = _ ,  
Roy = _ ,  
Res = [X,Roy,roy,1,2,3,4]
```

"בהינתן רשימת איברים צבועים בצבעים אדום לבן וכחול, סדר מחדש את הרשחמה כך שהאיברים האדומים יופיעו תחילה, אחריהם הלבנים ולבסוף הכחולים. הסדר הפנימי בתוך איברים מצבע מסוים ישמר"

דוגמת ריצה :

?- dutch([red(1),blue(2),red(3),blue(4),white(5)],RedsWhitesBlues).

RedsWhitesBlues = [red(1),red(3),white(5),blue(2),blue(4)]

dutch(Xs,RedsWhitesBlues):-

 distribute(Xs,Reds,Whites,Blues),
 conc(Whites,Blues,WhitesBlues),
 conc(Reds,WhitesBlues,RedsWhitesBlues).

distribute([**red(X)**|Xs],[**red(X)**|Reds],Whites,Blues):-
 distribute(Xs,Reds,Whites,Blues).

distribute([**white(X)**|Xs],Reds,[**white(X)**|Whites],Blues):-
 distribute(Xs,Reds,Whites,Blues).

distribute([**blue(X)**|Xs],Reds,Whites, [**blue(X)**|Blues]):-
 distribute(Xs,Reds,Whites,Blues).

distribute([],[],[],[]):-! .

פתרון שני , עם רשימות הפרש , חיסכון של שרשור :

dutch(Xs,**RedsWhitesBlues**):-

 distribute(Xs,**RedsWhitesBlues**-WhitesBlues,WhitesBlues-Blues,Blues-[]).

distribute([**red(X)**|Xs],[**red(X)**|Reds]-Tail,W-TW,B-TB):-

 distribute(Xs,Reds-Tail,W-TW,B-TB).

distribute([**white(X)**|Xs],R-TR,[**white(X)**|Whites]-Tail,B-TB):-

 distribute(Xs,R-TR,Whites-Tail,B-TB).

distribute([**blue(X)**|Xs],R-TR ,W-TW, [**blue(X)**|Blues]-Tail):-

 distribute(Xs,R-TR , W-TW ,Blues-Tail).

distribute([],R-R,W-W,B-B).

שאלה

הגדר את הפרדיקט `flatten_dl(L1, L2)`. `L1` היא רשימה שבה כל איבר הוא אטום או רשימה, ו-`L2` היא רשימת הפרש המייצגת את האיברים המופיעים ברשימה המקורית `L1` בסדר ממויין (וללא קינון). לצורך ההגדרה השתמש בהגדרת הפרדיקט `flatten` (המופיעה בעמוד 83 במדריך הלמידה), אך תקן הגדרה זו כך שייעשה שימוש ברשימות הפרש לצורך ייעול פעולת השרשור.

```
| ?- flatten_dl (f(d(a,d),aa) , A).
```

```
A = [f,d,a,d,aa|_31704] - _31704
```

```
| ?- flatten_dl(f(d(a,d),aa), A-[]).
```

```
A = [f,d,a,d,aa]
```

```
flatten_dl(Term,[Term|T]-T):-  
    atomic(Term),!  
    ;  
    var(Term),!.
```

```
flatten_dl(Term,[Functor|Rest]-T):-  
    Term=..[Functor|Args],  
    flatten_list(Args,Rest-T).
```

```
flatten_list([X|Xs],L1-T2):-  
    flatten_dl(X,L1-T1),  
    flatten_list(Xs,T1-T2).
```

```
flatten_list([],T-T).
```

נוכל לקבל רשימה נקייה אם נשתמש ביחס בצורה הבאה :

```
flatten(L,Res):-  
    flatten_dl(L,Res-[]).
```

```
| ?- flatten(f(d(a,d),aa), A).  
A = [f,d,a,d,aa]
```

מבנה הנתונים תור :

init_queue:-

retractall (queue (_)).

push(X):-

assertz(queue(X)).

pop(X):-

retract(queue(X)).

מבנה הנתונים תור :

Queue:

init_queue:-

retractall (queue (_)), assert(queue (L-L)).

push(X):-

retract(queue(L-[X|T])), assert(queue(L-T)).

pop(X):-

retract(queue(L-T)), L==T, !, assert(queue(T-T)), fail.

pop(X):-

retract(queue([X|L]-T)), assert(queue(L-T)).

מה התוכנית עושה ?

f([X1,X2|Xs],[Y|Ys],[Z|Zs]-T):-

E=..[Y,X1,X2],

Z is E,

Z >= 0,!,

f(Xs,Ys,Zs-T).

f([X1,X2|Xs],[Y|Ys],Zs-T):-

E=..[Y,X1,X2],

Z is E,

f(Xs,Ys,Zs-[Z|T]).

f([],[],T-T).

g(L1,L2,L3):-

f(L1,L2,L3-[]).

| ?- g([1, -2, 3, 4, 5, -1, 6, -2, 7, 3], [+, /, *, /, -], L).

מה התוכנית עושה ?

do(InpList, Item) :- f(InpList - [], Item).

f([Item|L] - L, Item) :- ! .

f([Item,X|L] - L, Item) :- ! .

f([X|A] - Z, Item) :-

g(B - C, _ , A - Z),

| ?- do([1, -2, 3, 4, 5, -1, 6], Res).

f(B - C , Item).

g(L1 - [Item|Z2] , Item, L1 - Z2).

רשימות הפרש ועצים :

החזרת רשימת עלים בעץ :

פתרון ראשון , ללא רשימות הפרש :

```
getLeaf(nil,[]).  
getLeaf(t(nil,X,nil),[X]):-!.  
getLeaf(t(L,X,R),List):-  
    getLeaf (R,ListL),  
    getLeaf (R,ListR),  
    conc(ListL,ListR,List).
```

החזרת רשימת עלים בעץ :

getLeaf(Tree, Leafs):- פתרון שני , עם רשימות הפרש :
getdivLeaf(Tree, Leafs-[]).

getdivLeaf(nil, T-T).

getdivLeaf(t(nil, X, nil), [X|T]-T):-!.

getdivLeaf(t(L, _, R), L-T):-

getdivLeaf(L, L1-T1),

getdivLeaf(R, L2-T2),

conc(L1-T1, L2-T2, L-T).

החזרת רשימת עלים בעץ :

getLeaf(Tree, Leafs):-

getdivLeaf(Tree, Leafs-[]).

פתרון שלישי

עם רשימות הפרש

ללא conc :

getdivLeaf(nil, T-T).

getdivLeaf(t(nil, X, nil), [X|T]-T):-!.

getdivLeaf(t(L,_,R), L1-T2):-

getdivLeaf(L, L1-L2),

getdivLeaf(R, L2-T2).

**כתוב פרדיקט המקבל עץ ורמה בעץ ומחזיר את רשימת הצמתים
ברמה מסוימת בעץ ע"פ סידורם משמאל לימין .**

nodes_of_level(t(L,X,R),0,[X]):-!.

nodes_of_level(nil,_,[]).

nodes_of_level(t(L,X,R),N,List):-

N1 is N - 1 ,

nodes_of_level(L,N1,List1),

nodes_of_level(R,N1,List2),

conc(List1,List2,List).

nodes_of_level(t(L,X,R),N,List-T):-

N>0,

N1 is N-1,

nodes_of_level(L,N1,List-List2),

nodes_of_level(R,N1,List2-T).

nodes_of_level(t(L,X,R),0,[X|T]-T).

nodes_of_level(nil,_,List-List).

/*

?-nodes_of_level(t(t(nil,2,nil),4,t(t(nil,5,nil),6,t(nil,8,nil))),2,List-[]).

List = [5,8] ;

***/**