

Дәріс 13

Тізімдер және жөндеу



Тізім – үтір арқылы жазылып, тік жақшаға алынған логикалық термдер тізбегі

Бос тізім – ішінде элементтері жоқ, ашылған және жабылған бос тік жақша.

Тізім мысалдары: [1, house, f(4)], [], [a, b, c].

Тізім екі бөлікке бөлінеді: **басы** және **соңы**. Басы – тізімнің бірінші элементтері, соңы – тізімнің соңы. Басы мен соңын бөлу үшін тік сызық қолданылады.

Бөлу мысалы:



Тізім	Басы	Құйрығы
[a b, c]	a	[b, c]
[a, b c]	a, b	[c]
[a, b, c]	a, b, c	[]



$[a, b, c] \text{ у } [a \mid [b, c]] \Leftrightarrow [a, b \mid [c]] \Leftrightarrow [a, b, c \mid]$

Бос тізімді басы мен құйрығына бөлуге болмайды!!!

Соңы – ол әрқашан тізім!!!

Тізім – Пролог бағдарламалау тілінің құрылымы, келесі жолмен жазылуы мүмкін:

(Басы, Құйрығы)

$[a, b, c] \Leftrightarrow .(a, .(b, .(c, [])))$

$[a] \Leftrightarrow .(a, [])$

«Элементтің тізімге кіруі». $\text{member}(\text{Elem}, [\text{Elem} \mid _])$.

$\text{member}(\text{Elem}, [\text{Head} \mid \text{Tail}]) :- \text{member}(\text{Elem}, \text{Tail})$.

Бағдарлама келесі былай оқылуы мүмкін: «Элемент тізім құрамында бар, ол тізімнің басында немесе құйрығында».

Пролог келесі сұраныстарға қалай жауап береді?

Трассировканы орындаңыз:

?- $\text{member}(a, [b, a, c])$.

?- $\text{member}(a, [b, a, a])$.

?- $\text{member}(a, [b, c, X])$.

?- $\text{member}(X, [a, b, c])$.

«екі тізімді біріктіру» бағдарламасы: $\text{conc}(\text{List1}, \text{List2}, \text{ResultList})$:

?- $\text{conc}([a, b], [c, d], [a, b, c, d])$.



Шешімі:

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

2. `conc([Head | Tail], L, [Head | NewTail]) :- conc(Tail, L, NewTail).`

Назар аударыңыз, бағдарламаның екінші жолы басқаша жазылуы мүмкін:

2. `conc([Head | Tail], L, ResultList) :- conc(Tail, L, NewTail), ResultList = [Head | NewTail].`

Conc (бір жолға) қолданып **member** бағдарламасын жазамыз:

`member1(X, L) :- conc(L1, [X | L2], L).`

Тізімге элементті қосу бағдарламасын жазамыз:

add(Item, SourceList, DestList)?

`add(X,L,[X|L]).`

Тізімнен элементті алу (жою) бағдарламасын жазамыз: **del(Item, SourceList, DestList).**

`del(X,[X|Tail],Tail).`

`del(X,[Y|Tail],[Y|Tail]) :- del(X,Tail,Tail).`

`del` функциясын сұранысқа тексерейік:

?- `del(X, [red, green, blue]).`

Тізімдегі элементтердің барлық кірулерін жою бағдарламасын жазамыз:

delAll(Item, SourceList, DestList).



Ауыстырымдарды генерациялау бағдарламасын жазу керек, қайту және қайталау кезінде тізім элементтерінің барлық мүмкін болатын ауыстырымдарын генерациялау керек.

Шешімі:

`permutation([], []).`

`permutation(L, [X | P]) :- del(X, L, L1), permutation(L1, P).`

Бұл қалай жұмыс істейді. Индукция базасы: бос тізімді ауыстыру – бұл бос тізім. Индукционды ауыстыру: L тізімінде элемент жойылады. (`del` функциясы бірінші элементті жоюдан бастайды, ары қарай тізбектеле жоя бастайды), соңынан тізімнің басына ауысады. Ауысу тізімнің соңынан басталады.

?– `permutation([red, green, blue], P).`

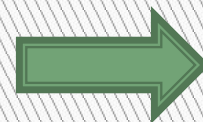
Тапсырма: сандар тізімі бар. Бұл тізімді қсу ретімен алу керек.

Шешімі:

`ordered([]).`

`ordered([_]).`

`ordered([X, Y | T]) :- X < Y, ordered([Y | T]).`



Бос тізім мен бір элементтен тұратын тізім – реттелген тізім. Егер тізімнің басында екі элемент бар болса, онда тізім реттелген егер бірінші элемент екіншісінен кіші болса онда құйрығы да реттелген.

Сұрыптау тапсырмасы бір жолмен шешіледі:

```
sort(SourceList, DestList) :- permutation(SourceList, DestList),  
                                ordered(DestList).
```

Сұрыптау әдісінде екі параметр бар: бірінші – бастапқы тізім, екіншісі – реттелген. Сұрыптау аяқталған болып есептеледі егер элементтерді ауыстырғаннан кейін бастапқы тізімде өсу ретімен реттелген тізім пайда болса.

«Маймыл мен банан» бағдарламасын қайта өңдейік, маймылдың бананға жету жолын есептеуді қосамыз.

1. start(1, 1).

2. stop(4, 4).

3. go(Path):- start(X, Y), move(X, Y, [], Path).

4. move(X, Y, P, [m(X, Y) | P]) :- stop(X, Y).

5. move(X, Y, From, To) :- X<5, X1 is X+1,

X1, Y, [m(X, Y) | From], To).

6. move(X, Y, From, To) :- Y<5, Y1 is Y+1,

Y, Y1, [m(X, Y) | From], To).



Бағдарламаның дұрыстығын тексеру үшін мына сұранысты орындаймыз:

?- go(P).

$P = [m(4,4), m(4,3), m(4,2), m(4,1), m(3,1), m(2,1), m(1,1)]$?;

$P = [m(4,4), m(4,3), m(4,2), m(3,2), m(3,1), m(2,1), m(1,1)]$?;

$P = [m(4,4), m(4,3), m(3,3), m(3,2), m(3,1), m(2,1), m(1,1)]$?

yes

Бұл мысалда бағдарлама екі рет дәлелденді бірінші рет сәтті дәлелденгеннен кейін.

Бұл қалай жұмыс істейді: move предикатына екі параметр қосылған. Бірінші қосылған параметр бос тізіммен қабылданады да, әрбір орындалған қадамды жинақтап отырады, екінші қосылған параметр – бананға жеткеннен кейін нәтижені рекурсивті түрде қайтарып отырады. Функтор $m(X, Y)$ біз келген нүктенің координаттарын сақтайды.

Жолдың кері тәртіппен берілетініне назар аударыңыз. Дұрыс тәртіппен қалай алуға болады?



1. start(1, 1).
2. stop(4, 4).
3. go(Path) :- start(X, Y), move(X, Y, [], Path).
4. move(X, Y, P, [m(X, Y) | P]) :- stop(X, Y).
5. move(X, Y, From, [m(X, Y) | To]) :- X < 5, X1 is X + 1, move(X1, Y, From, To).
6. move(X, Y, From, [m(X, Y) | To]) :- Y < 5, Y1 is Y + 1, move(X, Y1, From, To).

Сонда сұраныс нәтижесі басқаша болады:

?- go(P).

P = [m(1,1),m(2,1),m(3,1),m(4,1),m(4,2),m(4,3),m(4,4)]

yes

Бұл бағдарламалардың айырмашылықтарына назар аударыңыз.

Бірінші кезекте қолданылатын координаттарды рекурсияның соңына қарай қозғалған процессінде сақтаймыз, екіншісінде дәл осыны қайталаймыз бірақ рекурсиядан қайту жолында.



GBU Prolog та жөндеу: PISC

trace – предикат, трассировканы орныдаушы.

Сонымен бірге барлық бұйрықтар мен операциялар көрсетілетін болады: call, fail, exit, redo, exception.

notrace –трассировканы өшіру.

spy(бақыланушының спецификациясы) – белгілі бір предикатты жөндеу. Спецификациялар келесі түрде болады:

[PredSpec1, PredSpec2,...] –тізімдегі предикаттарды бақылау.

Name – бір предикатты бақылау.

Name/Arity – берілген аты бар және **арносты** предикатты бақылау.

Name/A1- A2 – A1 мен A2 аралығынан, берілген аты бар және **арносты** предикатты бақылау

nospy(Предикат атауы) – предикатты бақылауды өшіреді.

Қолдану мысалы:

?-spy(conc).

Spypoint placed on conc/3



conc/3).



Предикатты бақылау жұмыс істеу үшін жөндеудің келесі режимін қосу керек.

debug – жөндеу режимін қосады.

nodebug – жөндеу режимін өшіреді.

Жөндеу режимінде ақпараттың шығуын шектеу және трассировка үшін **leash** предикаты қолданылады (Баптау).

Бапту ретінде қолданылады:

full – эквивалентті [call, exit, redo, fail, exception]

half – эквивалентті [call, redo]

loose – эквивалентті [call]

none – эквивалентті []

tight – эквивалентті [call, redo, fail, exception]

?-leash(full).

Using leashing stopping at [call, exit, redo, fail, exception]

ports

Yes

Жөндеу қосылған трассировкамен дәлелдеу әрекеті болса бақылау қосылған debug предикатын дәлелдеу әрекеті автоматты түрде қосылады.



Трассировка процессінде / жөндеу келесі бұйрықтарды орындауға болады:

Enter – дәлелдеудің келесі жолына өту

a – дәлелдеуді тоқтату

h/? – жөндеу бұйрықтары жөнінде толық анықтама

Үй жұмысы:

Тізімнің ұзындығын есептеу бағдарламасын жазыңыз:

length(List, Length). Тізім соңының басына ену

бағдарламасын жазыңыз: **prefix(SubList, MainList).** Бір

тізімнің екіншісіне тізімнің соңы ретінде енің бағдарламасын

жазыңыз. **sublist(SubList, TestList).** Member

бағдарламасының аналогиясы жөнінде жазу керек.

length([], 0).

length([_ | Tail], N) :- length(Tail, N1), N is N1+1. prefix(X, Y)

:- conc(X, _, Y). sublist(S, L) :- conc(L1, L2, L), conc(S, L3, L2).

