Logic Programming

Section #4

Eman Safwat

31/10/2009

Lists (cont.)

Assignment Solution:

1. **even_list** ([1, 2, 3, 5, 6, 8], What)

What = [2, 6, 8]

```
even_list ( [ ], [ ]).

even_list ([H | T], [H | ET) :-
                    H mod 2 = 0,
                    even_list (T, ET).

even_list ([H  |  T], ET) :-
                    H mod 2 = 1,
                    even_list (T, ET).
```

هنا احنا عملنا 3 Rules أول واحدة ال Base Step:

```
even_list ( [ ], [ ]).
```

إنه ال List تكون فاضية فترجع List فاضية.

وتاني Rule للرقم لما يكون Even (وده اللي مفروض ال List ترجعوا):

```
even_list ([H | T], [H | ET]) :-
                    H mod 2 = 0,
                    even_list (T, ET).
```

الرقم بيدخل في ال Rule دي يت Check على ال Condition (H mod 2 = 0) لو الرقم Even هيقبل القسمة على 2 ويبقى ال Remainder ب Zero كده هترجع True، وهينده على نفسه ((even_list (T, ET) مديالله ال Tail ك Input parameter وبيرجع من ال Recursion ال Even Tail.

لو ملاحظين في ال Conclusion بتاعت ال Rule (even_list ([H | T], [H | ET]) ال H بترجع مع ال Output ليه بقى؟

عشان ال H بتحقق ال Condition (H mod 2 = 0) فبتبقى هيا ال Even Number اللي أنا بدور عليه، بعد مابينزل بال Levels ويوصل لل Base Step بيعمل Backtracking ويرجع كل مرة ال ET ويزود عليها ال H ويفضل يرجع كده لحد مايوصل لأول Recursion اتنده، وال H هتبقى أول Even Number في ال List.

لو في وسط ال List كده ظهر رقم Odd فال Condition (H mod 2 = 0) مش ه يتحقق وهيخرج من ال Rule ومش هيكمل Checking على باقي ال List، عشان كده عملنا Rule تالتة تعمل Handling لل Odd Numbers:

```
even_list ([H | T], ET) :-
                    H mod 2 = 1,
                    even_list (T, ET).
```

H mod 2 = 1 لما ال Remainder يبقى ب One الرقم هيكون Odd. بعدين بتنده على نفسها وتدي ال Tail ك Input وترجع ال ET ك Output زي تاني Rule عادي، الفرق إن هنا ال Output لل Rule في كل Level.. ET بس من غيرال H عشان هيا مش هتبقى Even Number مادام مابتحققش ال Condition.

Note:

- ممكن اخلي تاني Rule كده:

```
even_list ([H|T], [H|ET]) :-
                    H mod 2 = 0, !,
                    even_list (T, ET).
```

وأشيل ال Condition من تالت Rule فتبقى كده:

```
even_list ([H | T], ET) :-
                    even_list (T, ET).
```

زي ماعملنا في Last Section ال Cut هتمنع ال Backtracking لل Rule اللي بعدها لو كان الرقم Even، وبكده ال Speed بتاعت البرنامج هتزيد.

```
Get at (3, [5, 6, 11, 7, 8, 13], What)

What = 7

Get_at(0, [H|_], H) :- !.
Get_at (I, [H | T], X) :-
                    NI = I -1,
                    Get_at (NI, T, X).
```

هنا بيديني Index وعاوز ال Value بتاعته من ال List، احنا بنتعامل إنها Zero-Based.
هوا بياخد ال Index في ال Recursive Step (I) وينقص منه One ويبعت ال New Index (NI) مع ال Tail لنفسه تاني، كل مرة بيعمل كده لحد مال I تبقى ب Zero يدخل في ال Base Step ويرجع ال Element الباقي اللي هوا احنا عاوزينه.

من غير ال Cut إنه أما ال I توصل لل Base Step (I = 0) هتعمل Backtracking على ال Rule التانية وتبقي ب 1- وبعدين تدخل تاني في نفسها وتبقى ب 2- وكده لحد ماتوصل لل Empty List ساعتها هتقف عشان مش هيحصل Matching مع أي Rule من الاتنين وترجع ال Vlaue اللي أنا عاوزها، بس ال Cut هتمنع إنه ي Backtrack بعد مال (I = 0) فهتوفر عليا Time وMemory.

Example:

del (4, [1, 3, 4, 5, 7], What)
What = [1, 3, 5, 7]

بديله ال Value اللي عاوز اعملها Delete من ال List، ويرجعلي ال List بعد متشال ال Value.

```
del (H, [H | T], T).
del (X, [H | T], [H | NT]) :-
                    del (X, T, NT).
```

كل مرة بيشيل ال Head ويبعت ال Tail لنفسه لحد ما ألاقي ال Value اللي عاوزة اعملها Delete (H) كده وصلت ال Base Step وساعتها بأرجع ال T بس.

دايما بارجع ال H مع ال NT في ال Recursive Step كل مرة عشان ال Values كلها بتاعت ال List ترجعلي في الآخر، ماعدا ال Element اللي اتعمله Delete أكيد.

طيب لو ال Goal بقى كده:

Goal

del (4, [1, 4, 3, 4, 5, 4, 7], What).

كده هيطلع كذا Solution:

What=[1,3,4,5,4,7]
What=[1,4,3,5,4,7]
What=[1,4,3,4,5,7]
3 Solutions

ليه طلع ال Output ده؟
عشان هوا بعد ماهي Delete أول 4 ويطلع Solution هيعمل Backtracking علي ال Rule التانية ويعتبر ال 4 دي Head وياخد الباقي ك Tail ويعمل Delete لل 4 اللي بعدها ويطلع 4 اللي هيا ال

Head كجزء من ال Solution، وبعدين يي Backtrack علي ال Rule التانية ويعتبر ال Head 4 وويطلعها في ال Solution برضو وهكذا.. كل مرة يشيل 4 ويخلي ال Fours 2 التانيين!

عشان نمنع كل ده ونخليه يي Delete أول 4 بس بنحط Cut في أول Rule:

```
del (H, [H | T], T) :- !.
```

```
What=[1,3,4,5,4,7]
1 Solution
```

ليه طلع كده؟

عشان بعد ماعمل Delete لأول 4 جاي يعمل Backtracking لقي ال Cut فوقف ومادخلش ال Rule التانية وطلع ال Solution ده.

عايزين بقى ني Delete كل ال 4s على طول.. هنعمل كده:

```
del_all (_, [ ], [ ]).

del_all (H, [H | T], NT) :- !,
                    del_all (H, T, NT).

del_all (X, [H | T], [H | NT]) :-
                    del_all (X, T, NT).
```

أول Rule دي ال Base Step لو ال List فاضية رجع List فاضية، تاني Rule عشان لو ال X كانت Head وتالت واحدة عشان لو كانت في ال Tail.

دي زي Even List فلو فهمتوها كويس هتفهموا دي، الفرق بس إنه بي Delete أول مال H = H من غير Conditions زي التانية.

لو كتبت ال Goal ده في Del:

Goal

```
del (4, What, [1, 2, 3]).
```

بس وهيا من غير ال Cut هيطلعلي كده:

```
What=[4,1,2,3]
```

What=[1,4,2,3]

What=[1,2,4,3]

What=[1,2,3,4]

4 Solutions

إيه ال List اللي لو Delete منها ال 4 تديني [1, 2, 3] فبيطلعلي كل ال Combinations الممكنة من 4 مع ال List دي.

Tracing:

أول مرة بيدخل في ال Base Step:

H = 4, What = [H | T], T = [1, 2, 3]

What = [4, 1, 2, 3]

Backtrack

تاني Rule:

X = 4, What = [H |T], [H| NT] = [1, 2, 3]

يندە على نفسه بال Tail [2, 3] ويدخل في أول Rule:

H = 4, What = [H | T], T = [2, 3]

What = [4, 2, 3]

يرجع على تاني Rule بال T بتاعته ال هيا What (4, 2, 3) ويزود عليها ال H القديمة:

[H |T] = [1, 4, 2, 3]

What = [1, 4, 2, 3]

وبنفس الطريقة لحد مايطلع كل ال Solutions.

ممكن استعمل فكرة Delete دي مع Insert:

```
insert_all (X, L, NL) :-
                    del (X, NL, L).
```

Goal

```
insert_all (4, [1, 2], What).
```

الفكرة إنه بيجيب كل ال Possible Lists اللي لو اتشالت منها ال X هتبقى L. كإني ب Insert ال X كل مرة في ال L في مكان مختلف.

هيرجع كده:

What=[4,1,2]

What=[1,4,2]

What=[1,2,4]

3 Solutions

Example:

conc ([1, 2], [5, 6, 7], What).

What = [1, 2, 5, 6, 7]

اخدنا Conc **السكشن اللي فات إنها بتجمع** 2 Lists **على بعض. بتاخد** 2 Lists **ك** Input **وترجع** List.

طيب لو اديناها Goal **كده:**

conc (L1, L2, [1, 2, 3, 4]).

معني كده إنه عاوز 2 Lists **لو جمعتهم على بعض تديني ال** List [1, 2, 3, 4].

هيديني كل ال Combinations **الممكنة لل** 2 Lists.

L1=[], L2=[1,2,3,4]

L1=[1], L2=[2,3,4]

L1=[1,2], L2=[3,4]

L1=[1,2,3], L2=[4]

L1=[1,2,3,4], L2=[]

5 Solutions

نفس الفكرة لو استعملت Conc **عشان أعمل** Delete **لل** X:

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

بيقسم ال L **ل** 2 Lists **صغيرين ال** X **موجودة جوا واحدة منهم (بيخليها** Head **في تاني** List**)، وبعدين بياخد** L2 **اللي هيا** Tail **يعملها** Concatenate **مع** L1**، ويرجع** NL **اللي هيا** List **جديدة من غير ال** X.

Goal

del(5,[2, 5, 6, 7], What).

What=[2,6,7]

Example:

permute ([1, 2, 3], What).

عاوزين Permute دي تجيب ال Permutations للأرقام اللي في ال List.

شوفو ال Output الأول:

What=[1,2,3]

What=[2,1,3]

What=[2,3,1]

What=[1,3,2]

What=[3,1,2]

What=[3,2,1]

6 Solutions

هنعمل 2 Rules كالعادة واحدة Base Step والتانية Recursive Step، ال Base Step إنه ال List تكون Empty:

permute([ ], [ ]).

ودي ال Recursive Step:

permute ([H | T], NL) :-

permute (T, NT),

insert_all (H, NT, NL).

كل مرة بينده على نفسه بال Tail وبعدين يعمل Insert لل H في كل الأماكن مع ال Tail ده.

إزاي الكلام ده؟

Tracing:

permute ([1, 2, 3], What).

مع أول Rule مش هيعمل Matching عشان Empty List ودي مش Empty هيدخل في تاني Rule:

H = 1, T = [2, 3], NL = What

permute ([2, 3], NT)

أول Rule مش هت Match برضو، تاني Rule:

H = 2, T = [3], NL = NT

permute ([3], NT)

أول Rule مش هت Match، تاني Rule:

H = 3, T = [ ], NL = NT

permute ([ ], NT)

كده هيدخل في ال Base Step:

NT = [ ]

هيرجع لتاني Rule:

H = 3, NT = [ ]

insert_all (3, [ ], NL)

insert_all هتجيب كل ال Combinations بين 3 و [ ] اللي هيا:

NL = NT = [3]

هيرجع لل Level اللي قبله:

H = 2, NL = [3]

insert (2, [3])

كل ال Combinations اللي هما:

NL = [2, 3]

NL = [3, 2]

كل Solution من دول هيرجع لوحده (أول واحد هيتنفذ الأول ويطلع Solutions وبعدين التاني نفس الحكاية).

هيرجع لل Level اللي قبله:

H = 1, NL = [2, 3]

insert (1, [2, 3])

هترجع:

What=[1,2,3]

What=[2,1,3]

What=[2,3,1]

وبعدين ال Combination التانية:

H = 1, NL = [3, 2]

insert (1, [3, 2])

هترجع:

What=[1,3,2]

What=[3,1,2]

What=[3,2,1]

بس كده خلص البرنامج.

Compound Objects:

لحد دلوقتي كل اللي أخدناه كان Simple DataTypes (Atoms)، ال Lists بس اللي كانت Compound وكنت بعرّف ال Elements اللي جواها في Domains هنا برضو هاعمل كده.

ال Compound Objects دي زي ال Structs وال Classes في ال C++، بتبقى شايلة جواها كذا حاجة تانية. ولما آجي Access الحاجة دي لازم اندها باسم ال Struct.

وبديله حاجة اسمها Functor ده زي Variable يشيل ال Data اللي هتبقى جوا ال Object.

Example:

Domains

fullname = name (symbol, symbol).

هنا عرّف Data Type اسمه Fullname وادالله Functor .. name بيشيل 2 Symbols.

Predicates

nondeterm employee (fullname).

وبعدين قالو إن Employee هيبقى جواه Objects من نوع Fullname.

Clauses

employee (name (ahmed, hossam)).

employee (name (ali, hossam)).

employee (name (omar, sami)).

ودي Facts عادية زي اللي اخدناها في ال Simple Objects.

Goal

employee (Who).

Who=name("ahmed","hossam")

Who=name("ali","hossam")

Who=name("omar","sami")

3 Solutions

Who هنا رجعتلي كل حاجة عشان مفيش Constraints حطاها.

employee (fullname).

Employee دي مش بيعتبرها Predicate لأ بيعتبرها Object من نوع Fullname.

عشان كده أي حاجة جوا ال Employee لازم تبقى على شكل ال Fullname وال Values ت Match مع Fullname.

Goal

employee (name (Who, _)).

كده بقوله عاوزة أول Name في كل Employee أيان كان تاني Name، مش مهم بالنسبالي.

Who=ahmed

Who=ali

Who=omar

3 Solutions

employee (name (ahmed, _)).

yes

ده Yes Or No Question بيروح يعمل Matching مع ال Facts اللي في البرنامج ويرجع Yes لو كانت Matched مع حاجة. وطبعا أحمد هت Match.

Example:

ممكن نعمل Compound Object جواه Compound Object تاني زي كده:

Domains

fullname = name (symbol, symbol).

birthdate = date (integer, integer, integer).

student = stud (fullname, birthdate, real).

Predicates

nondeterm school (student).

Clauses

school (stud (name (ahmed, ali), date (12, 5, 1990), 90)).

school (stud (name (omar, hossam), date (15, 3, 1991), 90)).

school (stud (name (hossam, zaki), date (17, 3, 1990), 85)).

school (stud (name (sami, amr), date (1, 7, 1992), 88)).

school (stud (name (sami, ali), date (12, 5, 1990), 92)).

Goal

school (stud (name (Who, _), _, 90)).

Who=ahmed

Who=omar

2 Solutions

هنا هوا بيسأل على كل اللي جايبين 90 ومش مهم ال Last Name بتاعهم إيه ولا ال Birthdate.

school (stud (_, date (_, _, 1990), G)).

G=90

G=85

G=92

3 Solutions

هنا سأل عن ال Grade للناس اللي ال Birthdate بتاعه في سنة 1990 ومش مهم ال Month ولا ال Day وأي **Name** برضو مش هيفرق.

Example:

Domains

point = pnt (integer, integer).

line = ln (point, point).

rectangle = rect (point, point).

circle = cr (point, integer).

هنا أنا عرفت Point بتتحدد ب 2 integers هما ال x, y coordinates ليها.

وال Line وال Rectangle من 2 points، وال Circle من Point (Center) و integer (Radius).

Shape = pnt (integer, integer) ;

ln (point, point) ;

rect (point, point) ;

cr (point, integer).

Shape متعرف على إنه ممكن يبقى Point أو Line أو Rectangle أو Circle، ال Semi Colon ";" معناها OR.

Predicates

area (shape, real).

Clauses

area (pnt (_, _), 0).

area (ln (_, _), 0).

ال Area لل Point وال Line ب Zero.

area (rect (P1, P2), A) :-

P1 = pnt (X1, Y1),

P2 = pnt (X2, Y2),

W = abs (X1- X2),

H = abs (Y1 - Y2),

A = W * H.

Area ال Rectangle بحسبها بإني باخد ال Coordinates لأول Point وال Coordinates لتاني Point واحسب ال Width (X1- X2) وال Height (Y1 - Y2) وبعدين اضربهم في بعض تبقى دي ال Area.

abs دي Function جاهزة في ال Visual بتوجعلي ال Absolute Value.

area (cr (_, R), A) :-

A = (22/7) * R * R.

دي ال Area بتاعت ال Circle ايان كانت ال Point بكام، دايما باخد ال R اضربها في نفسها وفي ال π.

Goal

area (rect (pnt (10, 10), pnt (50, 60)), Area).

Area=2000

1 Solution

Good Luck