Languages and Algorithms for AI

Francesca Del Bonifro[1]

[1]Department of Computer Science and Engineering - UniBO

25/11/2019

Outline

Lists

Def. *append*/3
append([], L, L).
append([X|T1], L, [X|T2]) :- *append*(T1,L,T2).

Lists: reverse

Write a predicate *rev*/2 s.t *rev*(L1,L2) is true if L2 is the reverse of L1.

Lists: reverse

Write a predicate *rev*/2 s.t *rev*(L1,L2) is true if L2 is the reverse of L1.

rev([], []).

rev([H|T1], L) :- *rev*(T1, T), *append*(T, [H], L).

Queries:

?- *rev*([a,b,c],[c,b,a]).

?- *rev*([a,b,c],[c,a,b]).

?- *rev*([[a,b],c,d],X).

Lists: palindrome

Write a predicate *pal*/1 s.t *pal*(L) is true if L is palindrome.

Lists: palindrome

Write a predicate *pal*/1 s.t *pal*(L) is true if L is palindrome.
pal([]).
pal(R):-*rev*(R,R).
Queries:
?- *pal*([a,b,c,d]).
?- *pal*([a,b,c,c,b,a]).
?- *pal*([a,b,c,b,a]).
?- *pal*([[a,b],c,d,c,[b,a]]).

Lists: first element vs prefix

Write a predicate *first*/2 s.t. *first*(L,X) is true if X is the first element of the list L.

Lists: first element vs prefix

- Write a predicate *first*/2 s.t. *first*(L,X) is true if X is the first element of the list L.
- Write a predicate *prefix*/2 s.t. *prefix*(L,X) is true if X is a prefix of the list L.

Lists: first element vs prefix

Write a predicate *first*/2 s.t. *first*(L,X) is true if X is the first element of the list L.
first(L,X) :- *append*([X],_,L).
Write a predicate *prefix*/2 s.t. *prefix*(L,X) is true if X is a prefix of the list L.
prefix(L, X) :- *append*(X, _, L).
Queries:
?- *first*([a,b,c],X).
?- *prefix*([a,b,c],X).
?- *first*([[a,b],b,c],a).
?- *prefix*([[a,b],b,c],a).

Lists: last element vs suffix

- Write a predicate *last*/2 s.t. *last*(L,X) is true if X is the last element of the list L.
- Write a predicate *suffix*/2 s.t. *suffix*(L,X) is true if X is a suffix of the list L.

Lists: last element vs suffix

Write a predicate *last*/2 s.t. *last*(L,X) is true if X is the last element of the list L.
last(L, X) :- *append*(_, [X], L).
Write a predicate *suffix*/2 s.t. *suffix*(L,X) is true if X is a suffix of the list L.
suffix(L, X) :- *append*(_, X,L).
Queries:
?- *last*([a,b,c],X).
?- *suffix*([a,b,c],X).
?- *last*([a,[b,c]],c).
?- *suffix*([a,[b,c],c).

Lists: same list

Write a predicate *same*/2 s.t. *same*(L1,L2) is true if L1 and L2 represent the same list.

Lists: same list

Write a predicate *same*/2 s.t. *same*(L1,L2) is true if L1 and L2 represent the same list.

same([],[]).

same([X|T],R):-*append*([X],T,R).

Queries:

?- *same*([a,b,c],[a,f,c]).

?- *same*(X,[a,b,c]).

?- *same*([],X).

?- *same*([],[a]).

Train Ticket

How to represent these situations?
2x1 Special Offer: With the special offer 2x1 every Saturday you can travel in 2 paying only one Base ticket:

- The offer is valid for travel on all domestic trains at Business, Premium and Standard service levels, and in 1st and 2nd class;
- 2x1 does not apply to regional trains, Executive service level and couchette, VL and Excelsior services.

Train Ticket

Family Offer: for journeys by family groups made up of from 2 to 5 persons of whom at least 1 is an adult and 1 is a child of less than 12:

- 50% (30% for couchettes and VL) for children under 15;
- 20% for the other persons;
- The Family offer does not apply to journeys on the Excelsior service level.

Train Ticket

Eligibility rules:

- 2x1

eligibleFor(twoForOne, *ticketFor*(TrainCode, Day, ServiceLevel, Class,ListOfPassengersAndAge)) :-
train(TrainCode, Geo, SupportedServiceLevels, SupportedClasses),
member(ServiceLevel, SupportedServiceLevels),
member(Class, SupportedClasses) ,
Day = saturday ,
Geo \= regional ,
length(ListOfPassengersAndAge,2) ,
member(ServiceLevel, [business, premium, standard]) ,
\+ member(ServiceLevel, [executive, couchette, vl, excelsior]) ,
member(Class, [first, second]) .

Train Ticket

- Family

eligibleFor(family, *ticketFor*(TrainCode, Day, ServiceLevel, Class, ListOfPassengersAndAge)) :- train(TrainCode, Geo, SupportedServiceLevels, SupportedClasses) ,
member(ServiceLevel, SupportedServiceLevels) ,
member(Class, SupportedClasses) ,
length(ListOfPassengersAndAge,N) ,
N>=2 ,
N<=5 ,
member((_, Age1), ListOfPassengersAndAge) ,
Age1 >= 18 ,
member((_, Age2), ListOfPassengersAndAge) ,
Age2 < 12 ,
ServiceLevel \= excelsior .

Train Ticket

Ex.
train(676, ic, [business, premium, standard,couchette], [first, second]).
is a fact added to our knowledge base.
Queries:
?- *eligibleFor*(X, *ticketFor*(676, saturday, business, second,[(_,25),(_,10)])).
?- *eligibleFor*(X, *ticketFor*(676, saturday, standard, first,[(_,18),(_,20)])).
?- *eligibleFor*(X, *ticketFor*(676, monday, premium, second,[(_,18),(_,10)])).

train Ticket

Price rules:

- 2x1

computePrice(twoForOne, *ticketFor*(TrainCode, Day, ServiceLevel, Class, [_,Pass2]), Total) :-
computePrice(twoForOne, *ticketFor*(TrainCode, Day, ServiceLevel, Class, [Pass2]), Total) .
computePrice(twoForOne, *ticketFor*(TrainCode, _, ServiceLevel, Class, [_]), Total) :- price(TrainCode, ServiceLevel, Class, Total) .

train Ticket

- Family

computePrice(family, *ticketFor*(_,_, _, _, []), 0).
computePrice(family, *ticketFor*(TrainCode, Day, ServiceLevel,
Class, [(_,Age)|T]), Total) :-
price(TrainCode, ServiceLevel, Class, Price) ,
Age < 15 ,
\+ member(ServiceLevel, [couchette, vl]) ,
computePrice(family, *ticketFor*(TrainCode, Day, ServiceLevel,
Class, T), PartialPrice) ,
Discounted is Price*0.5 ,
Total is Discounted + PartialPrice .

Train Ticket

- Family

computePrice(family, *ticketFor*(TrainCode, Day, ServiceLevel, Class, [(_,Age)|T]), Total) :-
price(TrainCode, ServiceLevel, Class, Price) ,
Age < 15 ,
member(ServiceLevel, [couchette, vl]) ,
computePrice(family, *ticketFor*(TrainCode, Day, ServiceLevel, Class, T), PartialPrice) ,
Discounted is Price*0.7 ,
Total is Discounted + PartialPrice .

Train Ticket

- Family

computePrice(family, *ticketFor*(TrainCode, Day, ServiceLevel, Class, [(_,Age)|T]), Total) :-
price(TrainCode, ServiceLevel, Class, Price) ,
Age >= 15 ,
computePrice(family, *ticketFor*(TrainCode, Day, ServiceLevel, Class, T), PartialPrice) ,
Discounted is Price*0.8 ,
Total is Discounted + PartialPrice .

Train Ticket

Generally:
finalPrice(Offer,*ticketFor*(TrainCode, Day, ServiceLevel, Class, ListOfPassengersAndAges),Price):-
eligibleFor(Offer,*ticketFor*(TrainCode, Day, ServiceLevel, Class, ListOfPassengersAndAges)),
computePrice(Offer,*ticketFor*(TrainCode, Day, ServiceLevel, Class, ListOfPassengersAndAges),Price).

Train Ticket

For example, add to the knowledge base the price information for the train:

```
price(676, business, first, 100).
price(676, business, second, 80).
price(676, premium, first, 90).
price(676, premium, second, 60).
price(676, standard, first, 70).
price(676, standard, second, 50).
price(676, couchette, first, 130).
price(676, couchette, second, 110).
```

train Ticket

Queries:

?- finalPrice(family, *ticketFor*(676, monday, premium, second, [(_,80),(_,10),(_,19)]), X).

?- finalPrice(family, *ticketFor*(676, friday, couchette, first, [(_,80),(_,10),(_,19)]), X).

?- finalPrice(twoForOne, *ticketFor*(676, monday, business, first, [(_,30),(_,50)]), X).

?- finalPrice(twoForOne, *ticketFor*(676, saturday, business, first, [(_,30),(_,50)]), X).

?- finalPrice(Y, *ticketFor*(676, saturday, premium, second, [(_,10),(_,19)]), X).

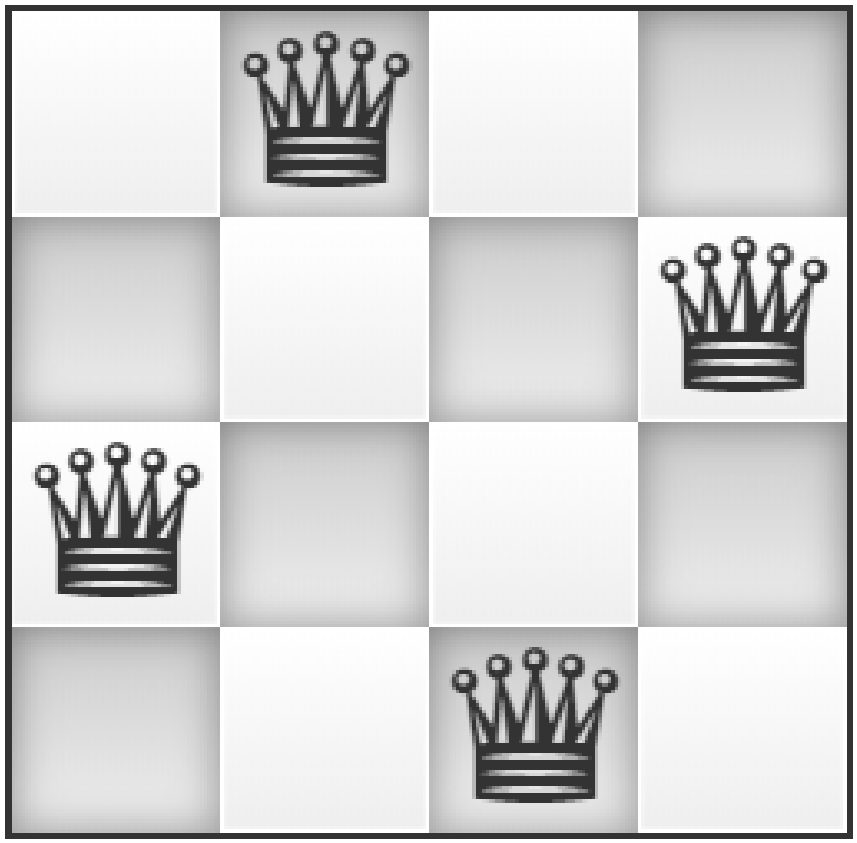
N-queens

Place N Queens on a NxN chess board s.t. no Queen can attack the others (in one move), i.e. no two queens are on the same row, column, or diagonal.
The solution Q is a list of length N. The i-th element of Q is the column number of the queen in the i-th row (the constraint s.t. no queen share the row number with the others is already satisfied).

N-queens

Es. N=4
Q=[2,4,1,3]
The plugin ’chess’ give us a visual representation of lists in this sense.

Q =



N-queens

```
:- use_rendering(chess).
queens(N,Queens) :- queens_list(N,Q), queens(Q,[],Queens).
queens_list(0,[]).
queens_list(N,[N|Ns]):- N>0, N1 is N-1, queens_list(N1,Ns).
queens([],Qs,Qs).
queens(U,P,Qs) :- select(Q, U,U1), no_attack(P,Q,1),
queens(U1,[Q|P],Qs).
no_attack([],_Q, _Nb).
no_attack([Y|Ys],Queen,Nb) :- Queen =\=Y+Nb,
Queen=\=Y-Nb, Nb1 is Nb+1, no_attack(Ys,Queen,Nb1).
```

N-queens

Queries:

?- queens(8, Q).

?- queens(20, Q).

?- queens(40, Q).

?- queens(100, Q).

Outline

Boolean constraints

We are in a village in which every inhabitant (variables) is either a knight or a knave.
knights always say the truth (true value for the correspondent variables)
knaves always lie (false value for the correspondent variables)

Boolean constraints

We meet A and B.
A says: "Either I am a knave or B is a knight."
:- use_module(library(clpb)).

Boolean constraints

We meet A and B.
A says: "Either I am a knave or B is a knight."
:- use_module(library(clpb)).
knights([A,B]) :- sat(A=:=($\sim$ *A* + B)).
Query:
?- *knights*(X).
The only solution is: X=[1,1], which means A=1, B=1.

Boolean constraints

We meet A and B.
A says: "I am a knave, but B isn’t."
:- use_module(library(clpb)).

Boolean constraints

We meet A and B.
A says: "I am a knave, but B isn't."
:- use_module(library(clpb)).
knights([A,B]) :- sat(A=:=($\sim$ *A* * B)).
Query:
?- *knights*(X).
The only solution is: A=0, B=0.

Boolean constraints

We meet A and B.
A says: "At least one of us is a knave."
:- use_module(library(clpb)).

Boolean constraints

We meet A and B.
A says: "At least one of us is a knave."
:- use_module(library(clpb)).
knights([A,B]) :- sat(A=:=card([1,2],[$\sim A$,$\sim B$])).
Query:
?- *knights*(X).
The only solution is: A=1, B=0.

Boolean constraints

We meet A, B and C.
A says: "All of us are knaves.". B says: "Exactly one of us is a knight."
:- use_module(library(clpb)).

Boolean constraints

We meet A, B and C.
A says: "All of us are knaves.". B says: "Exactly one of us is a knight."
:- use_module(library(clpb)).
knights([A,B,C]) :- sat(A=:=($\sim A * \sim B * \sim C$)),
sat(B=:=card([1],[A,B,C])).
Query:
?- *knights*(X).
The only solution is: A=0, B=1, C=0.

Boolean constraints

We meet A, B and C.
A says: "B is a knave.". B says: "A and C are of the same kind."
:- use_module(library(clpb)).

Boolean constraints

We meet A, B and C.
A says: "B is a knave.". B says: "A and C are of the same kind."
:- use_module(library(clpb)).
knights([A,B,C]) :- sat(A=:= $\sim$ *B*), sat(B=:=(A=:=C)).
Query:
?- *knights*(X).
There are two solutions:
A=0,B=1, C=0
or
A=1, B=0, C=0

N-queens

Place N Queens on a NxN chess board s.t. no Queen can attack the others (in one move), i.e. no two queens are on the same row, column, or diagonal.
The solution Q is a list of length N. The i-th element of Q is the column number of the queen in the i-th row (the constraint s.t. no queen share the row number with the others is already satisfied).

N-queens

```
:- use_rendering(chess).
:- use_module(library(clpfd)).
queens(N, Q) :- length(Q, N), Q ins 1..N, safe_queens(Q).
safe_queens([]).
safe_queens([Q1|Q]) :- safe_queens(Q, Q1, 1),
safe_queens(Q).
safe_queens([],_, _).
safe_queens([Q1|Q], Q0, D0) :-
Q0 # \= Q1,
abs(Q0 - Q1) # \= D0,
D1 #= D0 + 1,
safe_queens(Q, Q0, D1).
```

N-queens

Queries:

```
?- queens(8, Q), labeling([ff], Q).
?- queens(20, Q), labeling([ff], Q).
?- queens(40, Q), labeling([ff], Q).
?- queens(100, Q), labeling([ff], Q).
```