Explanation of Code For "auction"

Prolog is a declarative relational language (loosely) based on logic and there are no procedures or functions as such, instead of a program consisting of a set of declarative statements which describe a problem. To respond to user's query, the Prolog inference procedure uses these statements to derive logical consequences of these statements and returns "true" or "false".

Therefore, to achieve the function of selecting a set of bids in a combinatorial auction (***auction (+Items, +Bids, ?Selected)***), we need to build some "relations" among "+Items", "+Bids" and "?Selected", so that computers can based on these "relations" to find the answer to user's query.

Generally, my solution is: list all subsets of the set of bids made by buyers, sort the list consisting all the subsets in the order of total income, then try the first element of the sorted list and verify whether it contains bids made by same buyer and whether it can result in the sale of all items. Otherwise, try second most valuable subset until there is no result and returns false. This solution can meet all requirements **including extension 1 and extension 2**.

```
auction(Is,Bs,Ss) :-
    pickBestSubset(Bs,Ss),  % extension 1 - maximise return
    isNotRepeated(Ss),          % extension 2 - multiple bids
    isSoldOut(Is,Ss).
```

I use mainly three predicates to build the relation: pickBestSubset, isNotRepeated and isSoldOut.

1. First Part —— pickBestSubset (Bs, Ss)

```
%% find all subsets of one set , in the order of total return
pickBestSubset(Xs,BestSubset) :-
    bagof(Ys,isASubset(Xs,Ys),Xss),
    select([],Xss,Yss),
    mergeSort(Yss,Zss),
    maxReturn(Zss,BestSubset).
```

The first part of my solution is predicate pickBestSubset(Bs,Ss), which is used to pick the most valuable subset of a set of bids made by buyers . It mainly use "isASubset" ,"mergeSort" and "maxReturn" , which in turn is used to find one subset of a given set , sort a given set consisting of all subsets of a certain set and select the first element of this sorted set – the most valuable subset.

The statement "select ([], Xss, Yss)" is used to remove "[]" from subsets, which ensure "maxReturn" will not contain duplicate "[]" and save a few steps when looking for answers. The program still can give correct answer without this statement.

```
    % is an element one of the subsets?
    % e.g. isASubset([item(a,10)],[]) or ([item(a,10)],[item(a,10)])
isASubset([], []).
isASubset([X|Xs], [X|Ys]):-
  isASubset(Xs, Ys).
isASubset([_|Xs], Ys):-
  isASubset(Xs, Ys).

    % based on a sorted list, return the head of the list
    % e.g. maxReturn([[bid(_,_,10)],[bid(_,_,5)],[]],[bid(_,_,10)])
maxReturn([],[]).
maxReturn([Xs|_],Xs).
maxReturn([_|Xss],Ys) :-
    maxReturn(Xss,Ys).
```

2. Second Part —— isNotRepeated (Ss)

```
%% check whether result has same buyer
isNotRepeated([]).
isNotRepeated([bid(_,_,_)]).
isNotRepeated([bid(X,_,_),bid(Y,_,_)|Xs]) :-
    X \= Y,
    isNotRepeated([bid(Y,_,_)|Xs]).
```

The second part of my solution is predicate isNotRepeated(Ss) ,which filters out answers that have same buyer. This predicate allows buyers to make multiple bids as strict alternatives.

Without this predicate, the result can be as follows:

```
auction([item(a,10)],[bid(buyer_1,[item(a,5)],100),bid(buyer_1,[item(a,5)],100)],Ss).

Ss = [bid(buyer_1,[item(a,5)],100),bid(buyer_1,[item(a,5)],100)];
false
```

3. Third Part —— isSoldOut (Is, Ss)

```
%% check whether the sum of all elements in Ss is equal to Is
isSoldOut([I|Is],[S|Ss]) :-
    sum([I|Is],[S|Ss],Ls),
    permutation([I|Is],Ls).
```

The third part of my solution is predicate isSoldOut(Is,Bs) , which is used to check whether current subset produced by preceding predicates can result in the sale of all items.

```
    % take a list of items and a list of bids , get totoal sum
    % result : [item(a,0),item(b,5),item(c,10)...]
sum([],_,[]).
sum([item(X,_)|Xs],[bid(_,[item(Y,Z)|Ys],_)|Zs],[S|Ss]) :-
    sumOfOneItemInAllBids(item(X,_),[bid(_,[item(Y,Z)|Ys],_)|Zs],S),
    sum(Xs,[bid(_,[item(Y,Z)|Ys],_)|Zs],Ss).
```

Predicate sum takes a list of items and a list of bids and calculates the total amount of each item required by all bids. Generally, predicate sum work as follows:

1. Call predicate sumOfOneItemInAllBids , which, as the name suggests , will calculate add up the amount of one item (e.g. type_1) in all bids
2. When called , predicate sumOfOneItemInAllBids will then call predicate sumOfOneItemInOneBid , which will count the amount of an item in one bid
3. Recursively do the preceding steps

For example,

```
sum([item(a,5),item(b,10)],[bid(buyer_1,[item(a,2),item(b,3)],0),
                            bid(buyer_2,[item(a,2),item(b,6)],0)],Xs).

Xs = [item(a,4),item(b,9)];
false
```

Code of sumOfOneItemInAllBids :

```
    % take one item and a list of bids , get sum of this type in this list
    % result : item(a,10),no bid -> item(a,0)
sumOfOneItemInAllBids(item(X,_),[],item(X,0)).
sumOfOneItemInAllBids(item(X,_),[bid(_,[item(Y,Z)|Xs],_)|Ys],item(X,N)) :-
    sumOfOneItemInOneBid(item(X,_),bid(_,[item(Y,Z)|Xs],_),item(X,N1)),
    sumOfOneItemInAllBids(item(X,_),Ys,item(X,N2)),
    N is N1+N2.
```

Example result of sumOfOneItemInAllBids :

```
sumOfOneItemInAllBids(item(a,5),[bid(buyer_1,[item(a,2),item(b,3)],0),
                                 bid(buyer_2,[item(a,2),item(b,6)],0)],Xs).

Xs = [item(a,4)];
false

sumOfOneItemInAllBids(item(c,10),[bid(buyer_1,[item(a,2),item(b,3)],0),
                                  bid(buyer_2,[item(a,2),item(b,6)],0)],Xs).

Xs = item(c,0);
false
```

Code of sumOfOneItemInOneBid :

```
    % take one item type and one bid , get sum of this type in this bid
    % result : buy 5 of item a -> item(a,5) , no item in bid -> item(a,0)
sumOfOneItemInOneBid(item(X,_),bid(_,[],_),S):-
    S = item(X,0).
sumOfOneItemInOneBid(item(X,_),bid(_,[item(X,Y)|_],_),S):-
    S = item(X,Y).
sumOfOneItemInOneBid(item(X,_),bid(_,[item(Y,_)|Xs],_),S) :-
    X \= Y,
    sumOfOneItemInOneBid(item(X,_),bid(_,Xs,_),S).
```

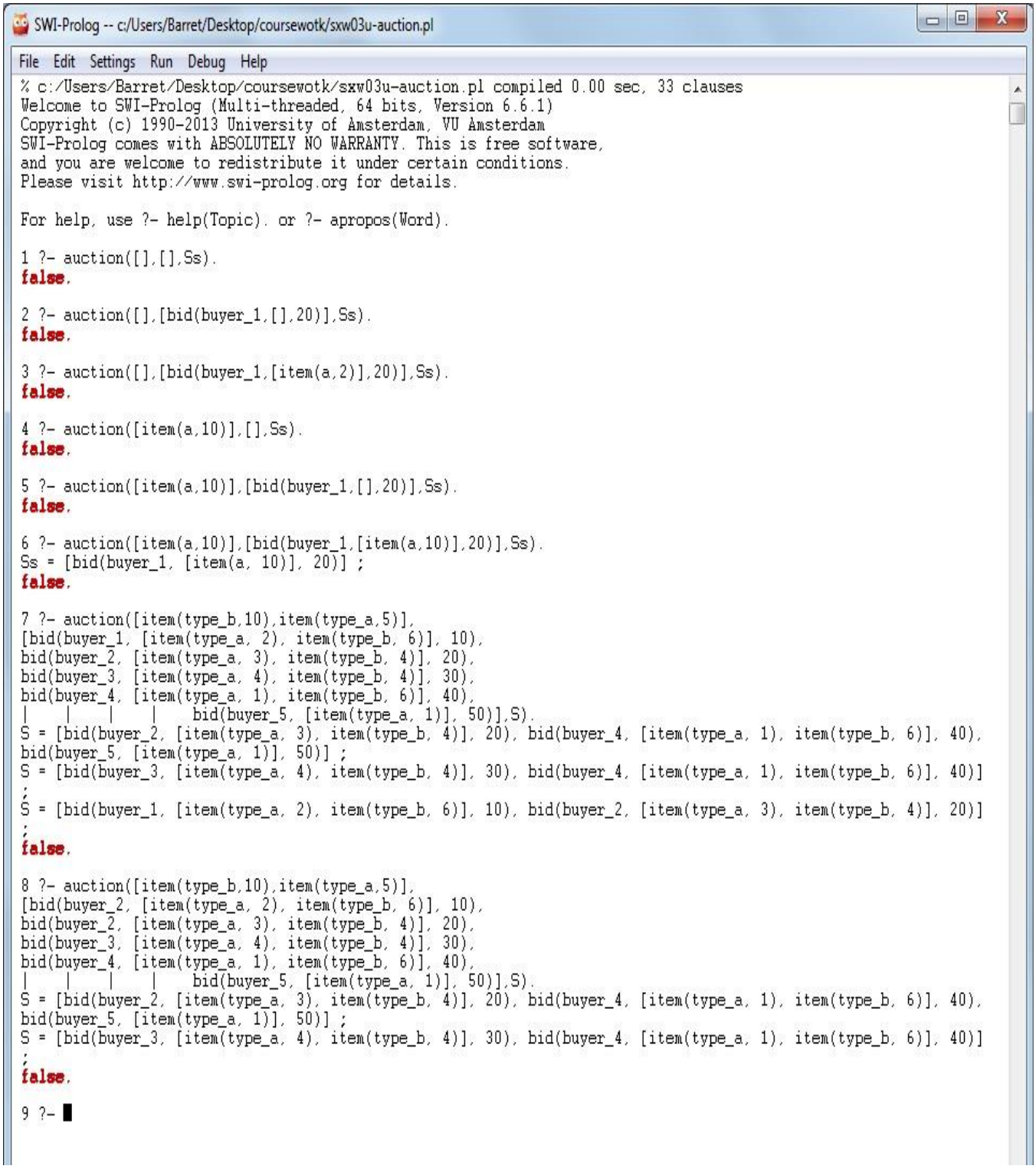
Example result of sumOfOneItemInOneBid :

```
sumOfOneItemInOneBid(item(a,5),bid(buyer_1,[item(a,2),item(b,3)],0),Xs).

Xs = [item(a,2)];
false

sumOfOneItemInOneBid(item(c,10),bid(buyer_1,[item(a,2),item(b,3)],0),Xs).

Xs = item(c,0);
false
```

Test

```
% test queries :
auction([],[],Ss).
Expected : Ss = false

auction([],[bid(buyer_1,[],20)],Ss).
Expected : Ss = false

auction([],[bid(buyer_1,[item(a,2)],20)],Ss).
Expected : Ss = false

auction([item(a,10)],[],Ss).
Expected : Ss = false

auction([item(a,10)],[bid(buyer_1,[],20)],Ss).
Expected : Ss = false

auction([item(a,10)],[bid(buyer_1,[item(a,10)],20)],Ss).
Expected : Ss = [bid(buyer_1,[item(a,10)],20)];
                false
```

```
auction([item(type_b,10),item(type_a,5)],
        [   bid(buyer_1, [item(type_a, 2), item(type_b, 6)], 10),
            bid(buyer_2, [item(type_a, 3), item(type_b, 4)], 20),
            bid(buyer_3, [item(type_a, 4), item(type_b, 4)], 30),
            bid(buyer_4, [item(type_a, 1), item(type_b, 6)], 40),
            bid(buyer_5, [item(type_a, 1)], 50)],S).
Expected : Ss = [bid(buyer_2, [item(type_a, 3), item(type_b, 4)], 20),
                 bid(buyer_4, [item(type_a, 1), item(type_b, 6)], 40),
                 bid(buyer_5, [item(type_a, 1)], 50)];
                 [bid(buyer_3, [item(type_a, 4), item(type_b, 4)], 30),
                 bid(buyer_4, [item(type_a, 1), item(type_b, 6)], 40)];
                 [bid(buyer_1, [item(type_a, 2), item(type_b, 6)], 10),
                 bid(buyer_2, [item(type_a, 3), item(type_b, 4)], 20)];
                 false

auction([item(type_b,10),item(type_a,5)],
        [   bid(buyer_2, [item(type_a, 2), item(type_b, 6)], 10),
            bid(buyer_2, [item(type_a, 3), item(type_b, 4)], 20),
            bid(buyer_3, [item(type_a, 4), item(type_b, 4)], 30),
            bid(buyer_4, [item(type_a, 1), item(type_b, 6)], 40),
            bid(buyer_5, [item(type_a, 1)], 50)],S).
Expected : Ss = [bid(buyer_2, [item(type_a, 3), item(type_b, 4)], 20),
                 bid(buyer_4, [item(type_a, 1), item(type_b, 6)], 40),
                 bid(buyer_5, [item(type_a, 1)], 50)];
                 [bid(buyer_3, [item(type_a, 4), item(type_b, 4)], 30),
                 bid(buyer_4, [item(type_a, 1), item(type_b, 6)], 40)];
                 false
```



The last two test queries show the general case of the program in use. The first query of these two can prove that **extension 1** is achieved and the second one can prove that **extension 2** is achieved, as compared to the first query, I only change buyer_1 to buyer_2 and it does return an expected result.