Declarative Programming with Constraints – Homework/Practice C1

General process for solving constraint satisfaction problems

1. Define the variables and their domains
2. Post the constraints
3. Perform labeling
4. If necessary, convert the values of the variables to the desired output format

Remember: there should be no branching (choice points) other than the ones created by labeling.

Problems

For each of the following problems, write a program using the SICStus CLP(FD) library that corresponds to the specification provided as a comment. Try to make your program efficient. In some cases (as stated in the text of the exercise), you are supposed to only post constraints but not perform labeling; this makes it easier to check that no choice points have been created.

Sample problems with solutions

1.
```
% send(?SEND, ?MORE, ?MONEY): each letter represents a digit, so that SEND+MORE=MONEY holds.
% The same letter represents the same digit, different letters encode different digits, and
% the leading digit of a multi-digit number must not be zero. send/3 should include labeling.
```

```
| ?- send(SEND, MORE, MONEY).
MORE = 1085, SEND = 9567, MONEY = 10652 ? ; no
```

   A possible solution:

```
:- use_module(library(clpfd)).

send(SEND, MORE, MONEY) :-
    Vars=[S,E,N,D,M,O,R,Y],
    domain(Vars, 0, 9), S#\=0, M#\=0,
    all_different(Vars),
    SEND#=1000*S+100*E+10*N+D,
    MORE#=1000*M+100*O+10*R+E,
    MONEY#=10000*M+1000*O+100*N+10*E+Y,
    SEND+MORE#=MONEY,
    labeling([], Vars).
```

2.
```
% partition(+L1, ?L2): L1 is a list of integers; L2 contains a subset of the elements of L1
% (in the same order as in L1), such that the sum of elements in L2 is half of the sum of
% elements in L1. partition/2 should include labeling.
```

   Hint: it is helpful to use n binary variables (where n denotes the number of elements of `L1`), with $x_i = 1$ meaning that the ith element of `L1` should also be an element of `L2` and $x_i = 0$ otherwise. It is fairly easy to formulate the constraint in terms of these variables. After labeling, do not forget to create the desired output based on the values of the x_i variables.

```
| ?- partition([1,2,3,5,8,13],L2).
L2 = [3,13] ? ;
L2 = [3,5,8] ? ;
L2 = [1,2,13] ? ;
L2 = [1,2,5,8] ? ;
no
```

   A possible solution:

```
partition(L1, L2) :-
    length(L1, N),
    length(Vars, N),
    domain(Vars, 0, 1),
    sum(L1, #=, S1),
    scalar_product(L1, Vars, #=, S2),
    S1#=2*S2,
    labeling([], Vars),
    create_output(L1, Vars, L2).

create_output([], [], []).
create_output([X|L1], [1|Vars], [X|L2]) :-
    create_output(L1, Vars, L2).
create_output([_|L1], [0|Vars], L2) :-
    create_output(L1, Vars, L2).
```

Problems to solve on your own

The solutions to the following problems must be submitted through ETS.

1.
```
% q(+A, ?B, ?C, ?D): A, B, C, D are positive integers, A=B+C+D, 2*B<C, 2*C<D. q/4 should not
% perform labeling.
```
```
| ?- q(14, B, C, D), labeling([], [B,C,D]).
B = 1, C = 3, D = 10 ? ;
B = 1, C = 4, D = 9 ? ;
no
```

2.
```
% sudoku_simple(?Matrix, +N): Matrix is an NxN matrix, containing integers between 1 and N.
% In each row and in each column, the numbers are pairwise different. sudoku_simple/2 should
% not perform labeling.
```
   The following example assumes that `library(lists)` has been loaded (because we use `append/2`).
```
| ?- Mx = [[3,1|_]|_], sudoku_simple(Mx, 3), append(Mx, _Vars), labeling([], _Vars).
Mx = [[3,1,2],[1,2,3],[2,3,1]] ? ;
Mx = [[3,1,2],[2,3,1],[1,2,3]] ? ;
no
```

3. In this problem, we are dealing with a generalization of the famous Fibonacci-series.
```
% fibonacci(+N, +Limit, ?L): N and Limit are positive integers; L is a list of length N, all
% elements of L are integers from the interval [1,Limit], and for any three consecutive
% elements A, B, C of L, the equation C=A+B holds. fibonacci/3 should not perform labeling.
```
```
| ?- fibonacci(5, 10, L), labeling([], L).
L = [1,1,2,3,5] ? ;
L = [1,2,3,5,8] ? ;
L = [2,1,3,4,7] ? ;
L = [2,2,4,6,10] ? ;
L = [3,1,4,5,9] ? ;
no
```

4. In the Knapsack problem, we are given n items; each item has a weight and a value. We have a knapsack with a given capacity. The aim is to select a subset of the items such that their total weight is not more than the capacity of the knapsack and their total value is at least a given number.
```
% knapsack(+Items, +MaxWeight, +MinValue, ?Selected): Items is a list of items of the form
% item(Id,Weight,Value), where Id, Weight, and Value are given positive integers; Id is a
% unique identifier of the item, whereas Weight and Value specify its weight and value.
% MaxWeight and MinValue are given positive integers specifying the capacity of the knapsack
% and the minimum total value of the items to be selected, respectively. Selected should be a
% list of the identifiers of the selected items, in the same order as they appear in Items.
% knapsack/4 should include labeling.
```
   Hint: it is helpful to use n binary variables, with $x_i = 1$ meaning that item i is selected and $x_i = 0$ meaning that item i is not selected. It is fairly easy to formulate the constraints in terms of these variables. After labeling, do not forget to create the desired output based on the values of the x_i variables.
```
| ?- knapsack([item(1,3,4),item(2,8,6),item(3,7,7),item(4,1,5)], 10, 10, Selected).
Selected = [3,4] ? ;
Selected = [2,4] ? ;
Selected = [1,3] ? ;
no
```