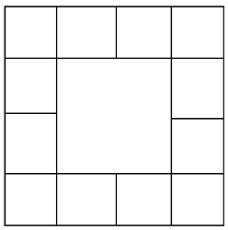
CLP in SICStus Exercise Sheet 2

Constraint Programming in SICStus

Goals:

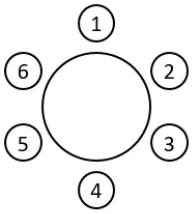
– Constraint Programming and Global Constraints

1. Guards in the Fort



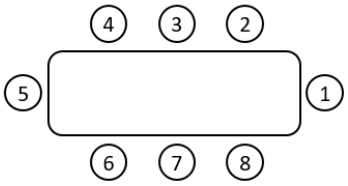
Twelve guards are tasked with guarding a square fort with four cells in each side. If a guard is posted in a side room, he can only see that side, while if he is posted in a corner room he can see both adjacent sides. Build a CLP predicate that can allocate the 12 guards in the fort in such a way that each side is guarded by exactly 5 guards.

2. Wedding Table



We want to sit six people (Adam, Bernadette, Christina, Diane, Francis and Emmet) around a table respecting some constraints:

- Adam and Bernadette should sit together
- Christina and Emmet should sit together
- Emmet and Francis should NOT sit together
- Adam and Emmet should NOT sit together



a) Solve this problem for a round table, trying to make the model flexible enough to adapt to different problem sizes. Try to improve performance by avoiding symmetries.

b) Consider now that the table is rectangular, with at least six places and an even total number of seats, where any two places in front of each are adjacent (e.g., places 3 and 7 in the figure, but not 1 and 5). Solve this problem using a model as flexible as possible and avoiding symmetries.

3. Another Car Line

Twelve cars are waiting in line at a traffic light. We know that:

- The cars have the following color distribution: 4 black, 2 green, 3 red and 3 blue;
- The first and last cars have the same color;
- The second and the penultimate cars have the same color;
- The fifth car is blue;
- All sequences of three consecutive cars have three distinct colors;
- Starting from the first car to the last, it is possible to see the sequence black-green-red-blue exactly once.

What is the color of each of the twelve cars in the line?

4. Exam Questions

When taking an exam, Emma wants to know which questions she needs to get right to obtain a given minimum grade. Implement ***questions(+Valuations, +MinGrade, -Right)***, that

receives the list of question values (integer values), the desired minimum grade, and returns in ***Right*** a list of binary variables representing the combination of wright (1) or wrong (0) answers allowing to obtain the desired minimum grade. Example:

```
| ?- questions([1, 2, 2, 3, 5, 1, 2, 4], 19, Right).
Right = [0,1,1,1,1,1,1,1] ? ;
Right = [1,1,1,1,1,0,1,1] ? ;
Right = [1,1,1,1,1,1,1,1] ? ;
no
```

5. Exam Preparation

Professor X needs to prepare an exam comprised of 7 questions, one for each of the seven topics in his course. He has a database containing several questions for each topic. Each question has a unique identifier, the topic identifier, a level of difficulty (from 1 to 3) and the expected time for the students to solve it (in minutes).

```
%question(IDQuestion, IDTopic, Difficulty, EstimatedTime).
question(1, 1, 1, 3).  question(2, 1, 2, 5).  question(3, 1, 3, 8).
question(4, 2, 1, 4).  question(5, 2, 2, 6).  question(6, 2, 3, 9).
question(7, 3, 1, 4).  question(8, 3, 2, 6).  question(9, 3, 3, 9).
question(10, 4, 1, 6). question(11, 4, 2, 9). question(12, 4, 3, 12).
question(13, 5, 1, 6). question(14, 5, 2, 9). question(15, 5, 3, 12).
question(16, 6, 1, 6). question(17, 6, 2, 9). question(18, 6, 3, 12).
question(19, 7, 1, 6). question(20, 7, 2, 9). question(21, 7, 3, 12).
```

The total time of the exam should be 55 minutes, and there should be at least two questions in the exam for each level of difficulty. Implement ***makeTest(+NCategories, +TotalTime, +MinQuestionsPerLevel, -Questions)*** that receives the number of desired categories, the desired total time and the minimum number of questions per difficulty level, retorning the chosen questions. Example:

```
| ?- makeTest(7, 55, 2, Q).
Q = [1,4,8,11,14,18,21] ?
```

6. Gift Wrapping

Gretchen has a number of presents to wrap and several rolls of wrapping paper, already open, with different patterns but with fixed width. To be properly wrapped, each present requires a given length of paper. Implement ***wrap(+Presents,+PaperRolls,-SelectedPaperRolls)***, which receives the list of presents ***Presents*** with the length of paper needed for each of them and the list of rolls ***PaperRolls*** with the available length in each paper roll, and returns in ***SelectedPaperRolls*** the paper rolls to use in each present.

Example:

```
| ?- wrap([12,50,14,8,10,90,24], [100,45,70], S).
S = [2,3,3,2,1,1,2] ? ;
S = [3,3,2,3,1,1,2] ? ;
no
```

7. Store Change

There is cash register for cash payments at a diner, that contains bills or coins of given values (e.g. 20€, 10€, 5€, 2€, 1€). When shopping, a client pays also using bills and coins, in a total amount equal or greater than the bill value. Implement ***change(+Coefs, +CashRegister, +Payment, +ToPay, -Change)***, obtaining in ***Change*** possible changes (amounts of bills and coins of each value). The register (***CashRegister***) is equally a list of amounts of bills and

coins of each type. ***Coefs*** contains the monetary value of each bill or coin – the indexes correspond to the ones in ***CashRegister*** and ***Change***. ***Payment*** contains the total amount paid by the client and ***ToPay*** is the total bill value (the change is the difference between these two). Example:

```
| ?- change([20,10,5,2,1], [4,5,1,1,0], 20, 13, Change).
Change = [0,0,1,1,0] ? ;
no
| ?- change([20,10,5,2,1], [4,5,2,1,0], 50, 13,Change).
Change = [0,3,1,1,0] ? ;
Change = [1,1,1,1,0] ? ;
no
```

8. Dance Pairs

A dance school needs to automatically pair their students. Given the heights of men and women in class (assumed to be in equal number), they require pairings where the difference of heights between man and woman is smaller than a given delta. The man can never be shorter than the woman. Implement ***dance_pairs(+MenHeights, +WomenHeights, +Delta, -Pairs)***, which receives the heights of men and women (lists of the same size) and the maximum height difference, returning in Pairs the pairings of people, identified by their indexes, and abiding by the constraints. Try to avoid symmetric solutions.

```
| ?- dance_pairs([95,78,67,84],[65,83,75,86],10,Pairs).
Pairs = [1-4,2-3,3-1,4-2] ? ;
no
| ?- dance_pairs([95,78,67,84],[65,77,75,86],10,Pairs).
Pairs = [1-4,2-2,3-1,4-3] ? ;
Pairs = [1-4,2-3,3-1,4-2] ? ;
no
```

9. Organizing Things

Sheldon needs to place N malleable objects, each with given weight (in grams) and dimensions (in cm3), in a cabinet with K compartments (shelves), in a matrix form. Each compartment k has an associated volumetric limit kv (in cm3). We to make sure no compartment has more weight than the one immediately below. Implement ***organize(+Shelves, +Objects, -Vars)*** receiving the volumetric limits of the compartments, as a list of lists, and Weight-Volume pairs for each object that determines possible object distributions, returning in ***Vars*** the allocation of objects to compartments. Example:

```
| ?- organize([[30,6],[75,15]], [176-40,396-24,474-35,250-8,149-5,479-
5], Vars).
Vars = [3,1,3,4,1,4] ? ;
Vars = [3,1,3,4,2,4] ? ;
no
```

10. Daily Calendar

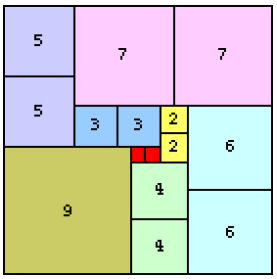
Carl wants to schedule some of his activities in such a way that some constraints are met:

- he always starts his day with two hours of sports;
- he wants to have a large breakfast, as well as lunch and dinner (each meal takes 1 hour);
- he needs to do his homework (duration: 2 hours), study for an upcoming exam (duration: 4 hours), and work on an assignment with an upcoming due date (duration: 3 hours);
- he likes to watch TV and read books, activities he does for 1 hour each, one of them being the last activity of the day;

- he wants the three meals to be in that order, separated by at least 4 hours of other activities, but by no more than 6 hours;
- he wants to have at least one hour between any two cerebral activities (like studying or doing homework) filled with other less mentally demanding activities.

Considering that Carl starts his day at 8h, and goes to bed at 24h, implement ***schedule(S)***, which gives possible schedules for his day.

11. Puzzle



Paul was given a puzzle to solve, but since he is a bit lazy to solve it by hand, he decides to make a CLP predicate to help. The puzzle consists in placing several smaller squares into a larger square, so that they all fit. The larger square is 19 units in size, and the smaller squares are distributed as follows: two squares each of sizes 1, 2, 3, 4, 5, 6, and 7, and one square of size 9. Implement ***squares(Pos)***, which provides the starting positions of each of the 15 smaller squares within the larger one.
Hint: order the list of smaller squares from largest to smallest.
Challenge: can you remove symmetries between pairs of same-sized squares?

12. 3-2-1 Sequence

Selena wants to generate a sequence of five numbers from 1 to 3, such that there is at least one occurrence of each, and the sequence is in non-ascending order.
Hint: solve this problem using a regular expression.