Exercise 1 (SLD Tree): (Not relevant for V3M and V3B! 4 points)
Consider the following Prolog program with constraints from $CT_{FD}$:

```
:- use_module(library(clpfd)).
fib(0, 1) :- !.
fib(1, 1) :- !.
fib(X, Y) :- A #= X - 1, B #= X - 2, fib(A, Y1), fib(B, Y2), Y #= Y1 + Y2.
```

A query of the form `fib(X, Y)` with $X \in \mathbb{N}$ instantiates the variable $Y$ with the $X$th fibonacci number. Please give a graphical representation of the full SLD tree for the query `?- X #>= 3, fib(X, Y)`. You can restrict your representation to those parts of the tree where the conjunction of constraints is satisfiable (i.e., do not show the parts where the edges contain unsatisfiable constraints).

Exercise 2 (Path Consistency): (Not relevant for V3M and V3B! 1 points)
Let $CO$ be a conjunction of constraints with variables $X_1, \ldots, X_n$. Let $a_1, \ldots, a_n \in \mathbb{Z}$. Please prove or disprove: $CT_{FD} \models CO[X_1/a_1, \ldots, X_n/a_n]$ implies $CO$ is path consistent.

Exercise 3 (Problem Solving with CLP): (Not relevant for V3M and V3B! 10 points)
The mathematical puzzle $n$-Sudoku is a generalization of the popular riddle Sudoku (which corresponds to 9-Sudoku in our setting). We are given a matrix of size $n \times n$. Each entry of the matrix can be a number from $\{1, \ldots, n\}$ or a variable. We call such a matrix a valid $n$-Sudoku if $n$ is a square number and if one can instantiate the variables in the matrix such that:

- Each row in the matrix must consist of $n$ different numbers.
- Each column in the matrix must consist of $n$ different numbers.
• One can partition the matrix into $n$ square blocks of size $\sqrt{n} \times \sqrt{n}$ each. Each of these $n$ blocks must consist of $n$ different numbers.

In Prolog we represent such a matrix via a list of its rows. Here a row entry can be either a number from $\{1, \ldots, n\}$ or a Prolog variable. For example, the matrix
\[
\begin{pmatrix}
1 & 2 & 3 & 4 \\
4 & X & 2 & 1 \\
3 & 1 & 4 & 2 \\
Y & 4 & 1 & 3
\end{pmatrix}
\]
is represented in Prolog via the term 
\[
[[1,2,3,4], [4,X,2,1], [3,1,4,2], [Y,4,1,3]].
\]

Your task is to implement a CLP predicate sudoku/2 in the constraint theory $CT_{FD}$. For a given matrix Board of numbers and variables, the predicate sudoku(Board, N) should be provable if Board can be instantiated to a valid $N$-Sudoku in our representation. In addition, if sudoku(Board, N) is provable, then variable entries in the matrix should be instantiated by the answer substitution such that the instantiated matrix is a valid $N$-Sudoku.

For example, the query
?- sudoku([[1,2,3,4], [4,X,2,1], [3,1,4,2], [Y,4,1,3]], 4).

should have the result $X = 3$, $Y = 2$.

Please complete the following code fragment by implementing the predicates occurring in the given declaration of sudoku/2.

% N-Sudoku is a generalization of Sudoku where the size of the
% (matrix-shaped) board is $N \times N$ and $N$ must be a square number.
% - Each cell of the board has entries from 1 to $N$.
% - Each row of the board has pairwise different entries.
% - Each column of the board has pairwise different entries.
% - The board is partitioned into $N$ squares of equal size.
% - Also each of these squares has pairwise different entries.
:- use_module(library(clpfd)).

% sudoku(Rows, Size) holds if Rows is a square Size x Size matrix
% which can be completed by an answer substitution to a legal
% Size-Sudoku.
sudoku(Rows, Size) :-
  Small_size #> 0, % each small square has size SmallSize
  Size #= Small_size * Small_size,
  legal_board(Rows, Size), % board must be a square matrix
  init_board(Rows, Size), % all its entries should take values in 1..Size
  % assert the three pairwise different conditions, too.
  rows_okay(Rows),
  columns_okay(Rows),
  squares_okay(Rows, Small_size, Size),
  label_board(Rows). % ask for a /concrete/ solution of the CLP

Hint: The pre-defined predicate transpose/2 from the constraint theory $CT_{FD}$ may be helpful in addition to the predicates from $CT_{FD}$ that you already know from the lecture. The atom transpose(Rows, Columns) holds if the matrix Columns and a transposed version of the input matrix Rows unify. For example, we have

?- transpose([[1,2], [X,4]], M).
M = [[1, X], [2, 4]].