Starting with Assignment 4

Step 1: Develop your solutions locally:

Include the CLP(FD) library as follows in your program:
:- use_module(library(clpfd)) .

Assignment 4.6. comes **in two parts**: implement a move6/2 procedure and the evalfinal/2 and slightly modify the eval/3 procedure of the tic-tac-toe game.

Furthermore, each of the 5 assignments 4.1., …, 4.5. comes **in two parts**:
Assignment 4.1. consists of move1/2 and lidm10/1,
Assignment 4.2. consists of move2/2 and **adfsmp20**/1,
Assignment 4.3. consists of move3/2 and lidm30/1,
Assignment 4.4. consists of move4/2 and lidm40/1,
Assignment 4.5. consists of move5/2 and lidm50/1,

That is, each assignment 4.X. needs two local tests, i.e. to test moveX/2 and lidmX0/1 (except for move2/2 and adfsmp20/1 for Assignment 4.2.).
And each assignment 4.X. needs two tests using tester4 (as outlined below).

Be aware that execution times of some queries, i.e. **lidm30(Path) and lidm(50) may last significantly longer** than in Assignment 2.

Like with Assignment 2 and 3, you shall first solve the queries locally by using the **SWI-Prolog-Interpreter -**
either by double-clicking the file a4.pl (or by opening it from command line, i.e.

```
using        swipl-win -s a4.pl    in Windows
or using     ./swipl -s a4.pl      in MacOS or Linux )
```

and the **SWI-Prolog-Editor .**

Step 2: Complete the assignments before you upload your solutions:

Be prepared that you can explain your solutions. Currently, it is not clear, which form of a presentation is possible. Nevertheless, you shall upload your solutions in time.
Upload your solution files not earlier than when you have done all assignments and commented your solutions as described below.

Before you upload your solutions, do a double check using the following tester4 program.
Be aware that execution times of the tester4 may last significantly longer than the execution times of your query in the interpreter!
The version of the test program differs depending on the operating system:

1. for Windows use **tester4**.exe
2. for MacOS use **tester4**
3. for OpenSuse Linux use **tester4suse**

1. Assuming that your program is called **a4.pl** and that you want to check your solution to Assignment 4.**1** (i.e. the farmer puzzle), in **Windows** use a CMD-Shell to call:

```
> tester4   a4   1                    for testing your procedure move1/2
> tester4   a4   10                   for testing your procedure lidm10/1
```

   (here, you shall use the Windows version tester**4**.exe)

2. Assuming that your program is called **a4.pl** and that you want to check your solution to Assignment 4.**1** (i.e. the farmer puzzle), in **MacOS** use a shell to call:

```
> ./tester4   a4   1                  for testing your procedure move1/2
> ./tester4   a4   10                 for testing your procedure lidm10/1
```

   (here, you shall use the MacOS version tester**4**)

3. Assuming that your program is called **a4.pl** and that you want to check your solution to Assignment 4.**1** (i.e. the farmer puzzle), in **OpenSuse Linux** use a shell to call:

```
> ./tester4suse   a4   1              for testing your procedure move1/2
> ./tester4suse   a4   10             for testing your procedure lidm10/1
```

   (here, you shall use the OpenSuse Linux version tester**4**suse)

Actions to be taken depending on the feedback of the program tester4:

Note that you need to have a correct solution to the moveX, before you should use tester4 for lidmX0, e.g. move1 should be correct, before you ask the tester4 to check lidm10, etc..

If you get an error message query X is undefined, make sure that you have named your procedures like in Assignment 4.X, i.e. move1, move2, …, lidm10, **adfsmp20**, lidm30 …, and that the number of parameters is correct.
If the program tester4 tells you that your solution is different from the wanted solution, look at the counter example presented to show the difference - and rewrite your query.
May be, you have to do this multiple times until you have a correct solution.

If the program tester4 tells you that your solution is most likely equivalent to the wanted solution, but it takes more clauses and/or more goals, try to improve your solution (i.e. find a shorter solution). May be, you have to do this multiple times until you have a short solution.
Hint: Try to find a correct solution in the first place. Once, your solution is equivalent to the wanted solution, but not short enough, try inlining (→ Video 1.10) to get a shorter solution. (If your solution is just one or two goals longer than the wanted solution tester4 does not mention this.)

If the program tester4 needs a very long runtime, for the execution time of your query X, make sure that query X executes rather fast locally in the SWI-Prolog interpreter. If this is not the case, you should try to speed-up your solution by checking the goal order (→ Video 2.4 and Quiz 2.4), by checking whether you stated all constraints correctly, and by checking

whether you have used the fastest strategies (→ videos 3.x) and whether you have used the CLP(FD) built-in predicates and the Prolog built-in predicates proposed in videos 3.x.
Furthermore, note that the tester4 needs a long time to check lidm30 and lidm50.
(With adfsmp the 4x3 puzzle would even need far too much run-time.)
Finally, if the tester4 needs a long time for query X0, doublecheck also your solution to query X!
Previous versions of the tester4 did not find all errors in query X that where the reason for exploding runtimes in queries X0.

If program tester4 tells you that your solution is most likely equivalent to the wanted solution, the tester could not detect any difference in the correctness of your program (and could not detect a difference of more than two goals in the complexity of the solutions).
Then you should prepare for explaining your solution.
Please do not upload your solution, before you get the answer your solution is most likely equivalent to the wanted solution **from the tester for each of the 12 checks**
> tester4 a4 N with N in {1,2,3,4,5,6,10,20,30,40,50,60} .

You are requested to write a short (3-4 lines) comment about each of your solutions to a procedure moveX below your code as part of the file that you upload. For example:

move1(…) :- < here is your implementation of the moves of the farmer puzzle >

% I prevent that the wolf eats the goat as follows …
% I avoided to see duplicate solutions by …

And you are requested to write a short (1 line) comment about your solutions to the procedure adfsmp20(…) below your code as part of the file that you upload. For example:

adfsmp(…) :- < here is your implementation of the paths of the brigde puzzle >

% I return only paths that keep the time limit by …

After commenting your solution, please check that it still compiles without warning (i.e. that it does NOT generate UTF-code warning messages)! Then, you can upload your solution.