UNIVERSITÄT BASEL

Lecturer: Thorsten Möller - **thorsten.moeller@unibas.ch**
Tutors: Miruna Muntean - **miruna.muntean@unibas.ch**
Mark Starzynski - **mark.starzynski@unibas.ch**

Programming Paradigms – Prolog FS 2023

Exercise 4 Due: 28.05.2023 23:55:00

Upload your answers to the questions **and source code** on Adam before the deadline.

Text : For answers to questions, observations and explanations, we suggest writing them in LaTeX. Please hand-in your answers as a **single PDF** file (independent of what tools you use, LaTeX, Markdown etc.).

Source-Code : For coding exercises, the source-code must be provided and has to be **commented in detail** (e.g. how it works, how it is executed, comments on conditions to be satisfied).

Upload : Please archive multiple files into a **single compressed zip-file**. If you upload an updated version of your solutions, the file name should contain a clear and intuitive versioning number. Only the latest version will be graded.

Requisit : In order to take the final exam, you must score at least $^2/_3$ of all available points throughout the mandatory exercises.

Modalities of work: The exercise can be completed in groups of at the most 2 people. Do not forget to provide the full name of all group members together with the submitted solution.

Question 1: Bite-sized Prolog Tasks (6 points)

Write Prolog facts and/or rules that complete each of the following tasks. Also include an example usage of the predicate that completes the task in a separate text file or as a comment in the code.

a) List-Manipulation

1. Define a list with elements 2, 4, 6, 8.

```
?- lst(X).
X = [2, 4, 6, 8].
```

2. Reverse a list but only keep the odd numbers.

```
?- rev([1, 2, 3, 4, 5], X).
X = [5, 3, 1].
```

3. Calculate the length of a list and return its cubed value.

```
?- listLength([1, 2, 3, 4], X).
X = 64.
```

4. Return and remove the last element of a list.

```
?- pop([1, 2, 3, 4], X, Y).
X = 4,
Y = [1, 2, 3].
```

(4 points)

b) Check if a number is prime.

```
?- isPrime(13).
true.
```

(2 points)

Question 2: Knowledge Base (5 points)

The following link contains information about canines, the living subfamily of canids (from latin canis, "dog"): https://www.wikiwand.com/en/Caninae

a) Write a Prolog knowledge base that contains at least 25 canines. Make sure to include all four divisions/tribes (see especially table at chapter "Phylogenetic relationships", you can neglect distinction between tribes/subtribes and just follow the table) and to differentiate between genus as well.

It should be possible to check if a canine is part of a given division and genus. The genus does not have to be part of the species name.

```
?- partOf(lycaon, Canis, Canina).
true.
```

It should also be possible to query the knowledge base for the division and genus of a given species or list every canine that is part of a division and/or genus.

```
?- partOf(cana, X, Y).
X = Vulpes,
Y = Vulpini.
```

(4 points)

b) Extend your knowledge base with a few of your favorite animals (living or extinct). For that possibly further classification is necessary:
e.g. lupus → Canis → Canina → Canidae → Mammalia → Animalia

(0 points)

c) Explain the concept of backtracking in Prolog. How is it responsible for finding a complete list of bones for a given body part and/or sub-structure?

(1 points)

Question 3: Unification (5 points)

What will Prolog return for the following terms? Briefly explain why each term can be unified or not.

1. `X.`
2. `[2, 8, X, 10] = [X|R].`
3. `3 - 1 = -(3, 1).`
4. `1 = [X|Y].`
5. `fun([a,b]) = fun([A, X|Y]).`

Question 4: 4x4 Sudoku (9 points)

4x4 Sudoku is a simplified version of sudoku (`https://en.wikipedia.org/wiki/Sudoku`), using only four 2x2 blocks of numbers instead of nine 3x3 blocks.

In this task you will write a Prolog program that can solve such simple Sudokus.

a) Write a predicate `piece/1` that checks whether a variable is a valid number for a 4x4 Sudoku (1-4).

(1 points)

b) Write a predicate `valid/1` that checks that a list contains no duplicate values.

(1 points)

c) Use your `piece` and `valid` predicates and write a predicate that can find a solution to a partially solved 4x4 sudoku and determine if a given solution is valid. Example:

```
?- simple([_,1,3,_,2,_,_,_,_,_,_,3,_,2,1,_],Row1,Row2,Row3,Row4).
Row1 = [4, 1, 3, 2],
Row2 = [2, 3, 4, 1],
Row3 = [1, 4, 2, 3],
Row4 = [3, 2, 1, 4] .
```

(7 points)

Question 5: Tic-Tac-Toe (Prolog) (BONUS) (0 points)

In this task - exactly same as you did in the sheets before with Python and Haskell - you have to implement the game Tic-Tac-Toe [1] on the command line, this time written **entirely** in the Prolog programming language.

Write a Prolog program that allows you to play the game on the command line.

Your program should have the following features:

- Starting player is chosen at random.
- If the board is filled completely without a winner, the game starts anew with the opposite player starting.
- Player can choose whether to play with another human player or against a computer AI.
- How you implement the AI is up to you, at its simplest you can make the computer play at random.
- Game conditions like restart, quit game, change mode (AI or 2nd player) should be implemented.
- Inputting the field of choice can be done via the numpad, e.g. for the lower left field you can press `1+ENTER` and for the middle field you press `5+ENTER` etc (sum of row and column number).

Game session start :

```
Turn of Player X
    1   2   3
  -------------
6 |   |   |   |
  ----+---+----
3 |   |   |   |
  ----+---+----
0 |   |   |   |
  -------------
```

Within a game session:

```
Turn of Player O
    1   2   3
  -------------
6 |   | X | O |
  ----+---+----
3 |   |   | X |
  ----+---+----
0 |   |   |   |
  -------------
```

You may implement only one of the two modes, but it is crucial that the console GUI and controls are the exact same as in your previous implementations and according to the provided description. Look at your Python/Haskell code and compare. Write down your opinion(s) about the comparison, what was easier for you, what more cumbersome, how do the implementations differ?

[1] **https://en.wikipedia.org/wiki/Tic-tac-toe**