12 Lab
Advanced Prolog, Java-Scala Prolog integration and a bit of planning

Mirko Viroli, Gianluca Aguzzi
{mirko.viroli,gianluca.aguzzi}@unibo.it

C.D.L. Magistrale in Ingegneria e Scienze Informatiche
Alma Mater Studiorum—Università di Bologna, Cesena

a.a. 2024/2025

Lab 12: Outline

- Meta-interpretation exercices
- Advanced Scala & Prolog exercises
  - Advanced foreach/monadic manipolation
  - A planner in pure Prolog
- `https://github.com/unibo-pps/pps-lab12`

Part 1: Meta-interpretation: check and understand

- consider the following generic map predicate

```
% map(+L, +Mapper, -Lo)
% where Mapper=mapper(I,O,UNARY_OP)
% e.g. Mapper=mapper(X, Y, Y is X+1)
map([], _, []).
map([H|T], M, [H2|T2]) :-
    map(T, M , T2), copy_term(M,mapper(H, H2, OP)), call(OP).
```

- and consider the following goal and result

```
?- map([10,20,30], mapper(X, Y, Y is X+1), L). -> L/[11,21,31]
```

- Explanation
  - in the recursive case, we first call recursively to the tail
  - then clone the mapper (to save its variables) providing the proper inputs, then call the operation
  - note: `copy_term(mapper(X,Y,Y is X+1),mapper(10,R,OP))` gives `OP/is(R, 10+1)`, hence `call(OP)` gives `R/11`

Part 2: Implement generic predicates

- Analogously to previous case implement some of the following predicates, using the "generic" higher-order approach of previous slide:
  - `filter` over lists
  - `reduce` over lists
  - `foldleft` over lists
  - `foldright` over lists
  - rewrite `map/filter` such that they reuse `foldright`
  - rewrite `reduce` such that it reuses `foldleft`

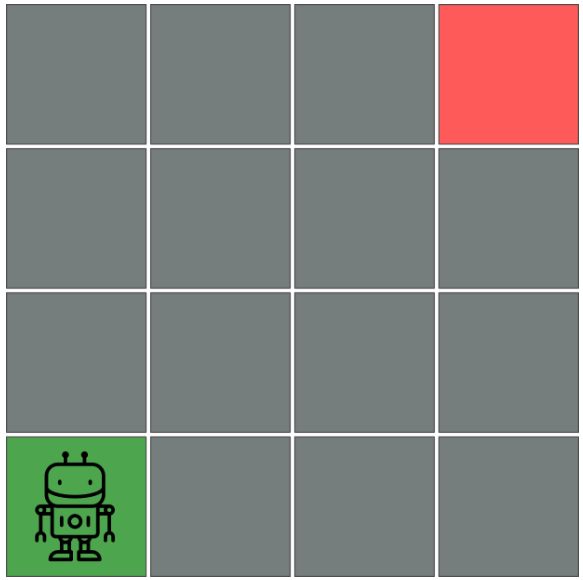
Part 3: Permutations

- In slides we saw how creating permutations of a list in Scala can be done by Java integration
- We now ask you to solve the same problem in pure Scala by a monadic/foreach approach
- Simple approach:
  - use for comprehension (recall n-queens problem)
  - use scala collections
- Suggestion:
  - start with the code proposed in `Permutation.scala`
  - be inspired by the example in slides: `lookup/3`

Part 4: A simple Prolog Planner

Problem

- Consider a 4x4 grid world enrivonment in which a robot livest
- Goal: starting from a given position find the list of actions that moves the robot towards the goal position (i.e. a plan)
  - actions: `up`, `down`, `left`, `right`



Part 4: A simple Prolog Planner

Prolog Formulation

- Position: A 2-arity predicate that contains x and y position of the grid world: `s(x, y)`
- Initial position (i.e. condition): a predicate that contains the robot initial position: `init(s(x, y))`
- Goal: a predicate with the desired final position: `goal(s(x, y))`
- Commands: atoms that identify the robot direction: `up`, `down`, `right`, `left`
- Movements: predicate that, given the robot an a direction, move it to the correspoding cell (if it is in the bound of the world): `move(D, s(X, Y), s(NX, NY))`

Part 4: A simple Prolog Planner

Ex 4.1: plan(+MaxN, -Trace)

```
1  init(s(0,0)) % initial condition
2  goal(s(3,3)) % goal
3
4  move(up, s(X, Y), s(X2, Y)) :- X>0, X2 is X-1
5  move(down, s(X, Y), s(X2, Y)) :- X<3, X2 is X+1
6  move(left, s(X, Y), s(X, Y2)) :- Y>0, Y2 is Y-1
7  move(right, s(X, Y), s(X, Y2)) :- Y<3, Y2 is Y+1
8  %plan(+MaxN, -Trace)
9  %produce the Trace (i.e. a list of commands)
10 %that, starting from (0,0)
11 %moves the robot to (3,3)
```

- plan should use move in order to create a list of commands (no longer than MaxN)

Part 4: A simple Prolog Planner

Ex 4.2: improve the game

- enjoy and add features to the environment (pitfalls destroying the robot, automatic jumps from a cell to another,. . .)
- enjoy and add actions (only once, the agent can jump two cells, . . .)

Ex 4.3: add visuals

- attach a GUI to see the robot actually moving