Prolog Examination (Recurso): L.EIC024-2024/2025

FEUP

2024/2025

Part I: Instructions and Database

Constraint: Answer questions 1 to 5 **WITHOUT** using multiple solution predicates (`findall/3`, `setof/3`, and `bagof/3`) and **WITHOUT** using any SICStus library.

```
% author(AuthorID, Name, YearOfBirth, CountryOfBirth).
author(1, 'John Grisham', 1955, 'USA').
author(2, 'Wilbur Smith', 1933, 'Zambia').
author(3, 'Stephen King', 1947, 'USA').
author(4, 'Michael Crichton', 1942, 'USA').

% book(Title, AuthorID, YearOfRelease, Pages, Genres).
book('The Firm', 1, 1991, 432, ['Legal thriller']).
book('The Client', 1, 1993, 422, ['Legal thriller']).
book('The Runaway Jury', 1, 1996, 414, ['Legal thriller']).
book('The Exchange', 1, 2023, 338, ['Legal thriller']).
book('Carrie', 3, 1974, 199, ['Horror']).
book('The Shining', 3, 1977, 447, ['Gothic novel', 'Horror', 'Psychological horror']).
book('Under the Dome', 3, 2009, 1074, ['Science fiction', 'Political']).
book('Doctor Sleep', 3, 2013, 531, ['Horror', 'Gothic', 'Dark fantasy']).
book('Jurassic Park', 4, 1990, 399, ['Science fiction']).
book('Prey', 4, 2002, 502, ['Science fiction', 'Techno-thriller', 'Horror', 'Nanopunk'
    ]).
book('Next', 4, 2006, 528, ['Science fiction', 'Techno-thriller', 'Satire']).
```

Pergunta 1

Implement `book_genre(?Title, ?Genre)`, which relates a book title to one of the genres of that book.

Pergunta 2

Implement `author_wrote_book_at_age(?Author, ?Title, ?Age)`, which unifies an author's name (`Author`) with the `Title` of a book they wrote and the `Age` of the author at the time the book was released. Consider the age of the author as the difference between the book's year of release and the author's year of birth. Example:

```
?- author_wrote_book_at_age('John Grisham', 'The Client', Age).
Age = 38 ?
```

Pergunta 3

Implement `youngest_author(?Author)`, which unifies `Author` with the name of the person who became an author at the youngest age. If more than one author published their first book at the same age, different results should be provided via backtracking. Example:

```
| ?- youngest_author(Author).
Author = 'Stephen King' ? ;
```

Pergunta 4

Implement `genres(+Title)`, which, without using recursion, prints all genres of the given book `Title` to the terminal (one per line, in the order by which they appear in the knowledge base). The predicate must always succeed. Solutions using recursion obtain at most 25% of the question's grade. Examples:

```
| ?- genres('The Firm').
Legal thriller
yes
| ?- genres('The Shining').
Gothic novel
Horror
Psychological horror
yes
```

Pergunta 5

Implement `filterArgs(+Term, +Indexes, ?NewTerm)`, which receives a Prolog Term as the first argument and a list of Indexes (starting at 1, sorted ascendingly), unifying the third argument with a new term containing only the arguments in the positions indicated by the indexes. Examples:

```
| ?- filterArgs(book('The Firm', 1, 1991, 432, ['L. thriller']), [1, 3, 5], NewTerm).
NewTerm = book('The Firm', 1991, ['L. thriller']).
| ?- filterArgs(author(1, 'John Grisham', 1955, 'USA'), [2, 3], NewTerm).
NewTerm = author('John Grisham', 1955).
```

Part II: Sets and Logic

Note: In the following questions, you can use set predicates (`findall/3`, `bagof/3`, and `setof/3`) as well as any SICStus library.

Pergunta 6

Implement `diverse_books(-Books)`, which returns a list of all the book titles with the greatest number of genres (if more than one exists any order is accepted). Example:

```
?- diverse_books(Books).
Books = ['Prey'] ? ;
no
```

Pergunta 7

Implement `country_authors(?Country, ?Authors)`, which unifies `Country` with a country of birth and `Authors` with the list of all authors born in that country. Example:

```
| ?- country_authors(Country, Authors).
Country = 'USA',
Authors = ['John Grisham','Stephen King','Michael Crichton'] ? ;
Country = 'Zambia',
Authors = ['Wilbur Smith'] ? ;
no
```

Pergunta 8

The Passionate Fans of Literature (PFL) Book Club keeps a record of all books read by its members, as exemplified in the following code:

```
read_book(bernardete, 'The Firm').
read_book(bernardete, 'The Client').
read_book(clarice, 'The Firm').
read_book(clarice, 'Carrie').
read_book(deirdre, 'The Firm').
read_book(deirdre, 'Next').
```

A book is considered popular if it has been read by at least 75% of all the members of the club. Implement `popular(?Title)`, which unifies `Title` with the title of a popular book. Example:

```
| ?- popular('The Client').
no
| ?- popular(Title).
Title = 'The Firm' ? ;
no
```

Pergunta 9

Consider the following code:

```
predX(_, [], [], []).
predX(R, [A|B], [C|D], [E|F]) :-
    S =.. [R, A, C, E],
    S,
    predX(R, B, D, F).
```

To which predicate of the `lists` library is `predX/4` equivalent?

A) `select/4`

B) `scanlist/4`

C) `cumlist/4`

D) `maplist/4`

E) `map_product/4`

Pergunta 10

Consider the existence of a `similarity(+Title1,+Title2,?Similarity)` predicate that returns a measure of similarity between two books, where Similarity is a value between 0 and 1, and the higher the value, the more similar the books are. Consider also the following implementation of a `most_similar(+Book,+OtherBook1,+OtherBook2,?MostSimilar)` predicate, which receives three book titles and unifies the fourth argument with either the second or third argument, whichever is more similar to the first book.

```
most_similar(Book, C1, C2, C1) :-
    similarity(Book, C1, S1),
    similarity(Book, C2, S2),
    S1 >= S2, !.
most_similar(Book, C1, C2, C2) :-
    similarity(Book, C1, S1),
    similarity(Book, C2, S2),
    S2 >= S1.
```

Which option is true regarding the cut in the code?

A) We require the implementation of `similarity/3` to determine whether the cut is red or green.

B) The cut can be considered red or green depending on the results of the call to `similarity/3`.

C) The cut is neither red nor green; it is unnecessary and can be safely removed.

D) The cut is red, as its removal would affect the results produced by `most_similar/4`

E) The cut is green, as its removal would not affect results but would affect performance.

Pergunta 11

Consider the existence of a `predZ/3` predicate with several clauses and the following code:

```
predZ(R, A, C):-
    T =.. [R, _, _],
    retract(( T :- S )),
    E =.. [R, A, C], E,
    asserta(( T :- S )).
```

What does a call to `predZ(predY, A, B)` do?

A) It replaces the arguments of the first clause of `predY/2` with the ones received by `predZ/3` as the second and third arguments.

B) It changes the order of the clauses of `predY/2`.

C) It changes the order of the goals in each clause of `predY/2`.

D) It simulates a call to `predY/2` without considering its first clause.

E) Nothing, any call to `predZ/3` will always fail.

Pergunta 12

We want to be able to write queries and code in the format 'Author wrote Book at Age'. Example:

```
?- Author wrote 'The Firm' at Age.
Author = 'John Grisham',
Age = 36 ?
```

Declare the necessary operators and code to allow queries in the format `Author wrote Book at Age`. In a parse tree, the `wrote` operator should be above the `at` operator.

Part III: The Codex Puzzle

Consider a codex, like the one shown in the figure, represented by a list of lists, as shown below. Consider that all the lists have the same length.

Figure 1: Codex

```
[ [a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z],
  [p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o],
  [p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o],
  [l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d, e, f, g, h, i, j, k],
  [e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, a, b, c, d] ]
```

Pergunta 13

Implement `rotate(+List, +Position, +Rotations, ?NewList)`, which receives a List with the initial configuration of the codex, the Position of the dial to rotate (consider that indexes start at 1), and the number of `Rotations` to be performed, unifying `NewList` with the resulting codex configuration. The predicate should fail if the received `Position` is outside the valid range. Both negative numbers and numbers larger than the size of the dial can be accepted. Example:

```
| ?- rotate([ [3,6,8], [5,0,7], [2,1,4] ], 2, 1, NewList).
NewList = [ [3,6,8], [0,7,5], [2,1,4] ]

| ?- rotate([ [3,6,8], [5,0,7], [2,1,4] ], 3, 2, NewList).
NewList = [ [3,6,8], [5,0,7], [4,2,1] ]

| ?- rotate([ [3,6,8], [5,0,7], [2,1,4] ], 1, -1, NewList).
NewList = [ [8,3,6], [5,0,7], [2,1,4] ]
```

```
| ?- rotate([ [3,6,8], [5,0,7], [2,1,4] ], 1, 4, NewList).
NewList = [ [6,8,3], [5,0,7], [2,1,4] ]
```

Pergunta 14

Implement `matches(+List, +Code)`, which succeeds if the codex `List` is aligned with `Code` as a solution (i.e., each digit of `Code` corresponds to the first digit of the list in the corresponding position of the codex), and fails otherwise. The predicate must fail if `Code` is different in size from the number of elements in the codex `List`. Example:

```
| ?- matches([ [3,6,8], [5,0,7], [2,1,4] ], [3,5,2]).
yes

| ?- matches([ [3,6,8], [5,0,7], [2,1,4] ], [6,0,1]).
no

| ?- matches([ [3,6,8], [5,0,7], [2,1,4] ], [3,5,2,1]).
no
```

Pergunta 15

Implement `move(+Initial, -Final)`, which returns in `Final`, one by one via backtracking, all the possible moves for the given `Initial` codex configuration (possible rotations for each of the dials). Example:

```
| ?- move([[3,6,8],[5,0,7],[2,1,4]], Final).
Final = [[6,8,3],[5,0,7],[2,1,4]] ? n
Final = [[8,3,6],[5,0,7],[2,1,4]] ? n
Final = [[3,6,8],[0,7,5],[2,1,4]] ? n
Final = [[3,6,8],[7,5,0],[2,1,4]] ? n
Final = [[3,6,8],[5,0,7],[1,4,2]] ? n
Final = [[3,6,8],[5,0,7],[4,2,1]] ? n
no
```

Pergunta 16

Implement `solve(+Code, +Key, -States)`, which returns in `States` the set of states through which the codex passes to be solved, avoiding repeated states (`States` must contain the initial `Code`, the intermediate ones, and the final one). `Key` contains the code (representing the first element of each list of the solution). Solutions that return shorter paths will be valued. Examples:

```
| ?- solve([[3,6,8], [5,0,7], [2,1,4]], [6,0,4], States).
States = [[[3,6,8], [5,0,7], [2,1,4]], [[6,8,3], [5,0,7], [2,1,4]],
        [[6,8,3], [0,7,5], [2,1,4]], [[6,8,3], [0,7,5], [4,2,1]]] ?
yes

| ?- solve([[3,6,8], [5,0,7], [2,1,4]], [6,5,1], States).
States = [[[3,6,8], [5,0,7], [2,1,4]], [[6,8,3], [5,0,7],
        [2,1,4]], [[6,8,3], [5,0,7], [1,4,2]]] ?
yes
```