Middle East Technical University | Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring 2021-2022

Programming Exam 8

Due date: 11 June, 2022, Friday, 11:59

1 Problem Definition

In this exam, you are going to implement predicates that operate on a knowledge base with movies.

1.1 Knowledge Base Predicate: `movie`

The `movie` predicate is used to define movies and their attributes. Its format is given below, where `Title` is the name of the movie, `Director` is the director of the movie, `ReleaseYear` is the year that the movie is released, `OscarNoms, EmmyNoms,` and `GoldenGlobeNoms` are nominations for Oscar, Emmy and Golden Globe awards respectively:

```
movie(Title, Director, ReleaseYear, OscarNoms, EmmyNoms, GoldenGlobeNoms).
```

The example knowledge base given to you in kb.pl file can be seen below:

```
movie(moonlight, barry_jenkins, 2016, 8, 0, 6).
movie(inside, bo_burnham, 2021, 0, 6, 0).
movie(parasite, bong_joon_ho, 2019, 6, 0, 3).
movie(gone_girl, david_fincher, 2014, 1, 0, 4).
movie(fight_club, david_fincher, 1999, 1, 0, 0).
movie(social_network, david_fincher, 2010, 7, 0, 6).
movie(lady_bird, greta_gerwig, 2017, 5, 0, 4).
movie(little_women, greta_gerwig, 2019, 6, 0, 2).
movie(dead_poets_society, peter_weir, 1989, 4, 0, 4).
movie(coda, sian_heder, 2021, 3, 0, 2).
movie(jojo_rabbit, taika_waititi, 2019, 4, 0, 2).
movie(what_we_do_in_the_shadows, taika_waititi, 2014, 0, 0, 0).
movie(matrix, wachowski_sisters, 1999, 4, 0, 0).
movie(matrix_reloaded, wachowski_sisters, 2003, 0, 0, 0).
movie(matrix_revolutions, wachowski_sisters, 2003, 0, 0, 0).
movie(matrix_resurrections, wachowski_sisters, 2021, 0, 0, 0).
movie(grand_budapest_hotel, wes_anderson, 2014, 9, 0, 4).
movie(fantastic_mr_fox, wes_anderson, 2009, 2, 0, 1).
movie(isle_of_dogs, wes_anderson, 2018, 2, 0, 2).
movie(moonrise_kingdom, wes_anderson, 2012, 1, 0, 1).
movie(french_dispatch, wes_anderson, 2021, 0, 0, 1).
```

2 Specifications

In this exam, you are expected to implement 5 different predicates with varying difficulties. All predicates will be queried with different arguments given as variables, but not all arguments will be queried for all predicates. Querying details for each predicate can be seen in the example queries for that predicate.

2.1 `movie_directed_by/2` - 10 points

Two argument predicate where `Title` is the name of the movie, `Director` is the name of the director. Predicate has the form:

```
movie_directed_by(Title, Director).
```

Example queries:

```
?- movie_directed_by(matrix, wachowski_sisters).
true.

?- movie_directed_by(matrix, Director).
Director = wachowski_sisters.

?- movie_directed_by(Title, wachowski_sisters).
Title = matrix ;
Title = matrix_reloaded ;
Title = matrix_revolutions ;
Title = matrix_resurrections.
```

- In the case of several possible answers for the `Title` variable for a given director, possible answers should be listed in the order of appearance in the knowledge base, which is the default behaviour of Prolog.

2.2 `total_awards_nominated/2` - 20 points

Two argument predicate where `Title` is the name of the movie, `Nominations` is the number of total award nominations. Predicate has the form:

```
total_awards_nominated(Title, Nominations).
```

Example queries:

```
?- total_awards_nominated(inside, Nominations).
Nominations = 6.

?- total_awards_nominated(what_we_do_in_the_shadows, Nominations).
Nominations = 0.
```

- This predicate **will not be queried with `Title` argument as a variable**, so you do not need to worry about the behavior of the queries of the form `total_awards_nominated(Title, 6)`.

2.3 `all_movies_directed_by/2` - 20 points

Two argument predicate where `Director` is the name of the director, and `Movies` is a list of movie titles. Predicate has the form:

```
all_movies_directed_by(Director, Movies).
```

Example queries:

```
?- all_movies_directed_by(wachowski_sisters, Movies).
Movies = [matrix, matrix_reloaded, matrix_revolutions, matrix_resurrections].
```

- This predicate **will not be queried with `Director` argument as a variable**, so you do not need to worry about the behavior of the queries of the form `all_movies_directed_by(Director,[coda])`.

Hint: `findall/3` base predicate may be helpful for implementing this predicate.

2.4 `total_movies_released_in/3` - 25 points

Three argument predicate where `Movies` is a list of movie titles, `Year` is the release year of movies, and `Count` is the count of movies that are released in the year `Year` from the list of titles `Movies`. Predicate has the form:

```
total_movies_released_in(Movies, Year, Count).
```

Example queries:

```
?- total_movies_released_in([inside,parasite,coda,matrix_resurrections,↩
    french_dispatch], 2021, Count).
Count = 4 .
```

- This predicate **will not be queried with `Movies` or `Year` argument as a variable**, so you do not need to worry about the behavior of the queries such as `total_movies_released_in(Movies, 2021, 4)` or `total_movies_released_in([inside,parasite], Year, 4)`.

2.5 `all_movies_released_between/4` - 25 points

Four argument predicate where `Movies` is a list of movie titles, `MinYear` and `MaxYear` are the integers to define the inclusive interval [MinYear, MaxYear], and `MoviesBetweenGivenYears` is a list of movie titles that are in the given list `Movies` and relased in the interval. Predicate has the form:

```
all_movies_released_between(Movies, MinYear, MaxYear, MoviesBetweenGivenYears).
```

Example queries:

```
?- all_movies_released_between([inside,parasite,social_network,little_women,↩
    dead_poets_society,fantastic_mr_fox], 2010, 2019, MoviesBetweenGivenYears).
MoviesBetweenGivenYears = [parasite, social_network, little_women].

?- all_movies_released_between([moonlight, coda], 1940, 1945, ↩
    MoviesBetweenGivenYears).
MoviesBetweenGivenYears = [].
```

- This predicate **will not be queried with `Movies`, `MinYear` or `MaxYear` arguments as a variables**, so you do not need to worry about the behavior of such queries.

3 Regulations

- **Implementation and Submission:** The template files are available in the Virtual Programming Lab (VPL) activity called "PE8" on ODTUCLASS. At this point, you have two options:
  - You can download the template files, complete the implementation and test it with the given sample I/O on your local machine. Then submit the same file through this activity.
  - You can directly use the editor of VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submitting a file.

  Please make sure that your code runs on ODTUCLASS. There is no limitation in running your code online. The last save/submission will determine your final grade.

- **Programming Language:** You must code your program in Prolog. Your submission will be run with swipl on ODTUCLASS. You are expected make sure your code runs successfully with swipl on ODTUCLASS.

- **Modules:** You are not allowed to import any modules. However, you are allowed to use the base predicates present in Prolog or define your own predicates.

- **Cheating: We have zero tolerance policy for cheating.** People involved in cheating (any kind of code sharing and codes taken from internet included) will be punished according to the university regulations.

- **Evaluation:** Your program will be evaluated automatically using "black-box" testing, so make sure to obey the specifications. No erroneous input will be used. Therefore, you don't have to worry about invalid cases.

  Important Note: The given sample I/O's are only to ease your debugging process and NOT official. Furthermore, it is not guaranteed that they cover all the cases of required functions. As a programmer, it is your responsibility to consider such extreme cases for the functions. Your implementations will be evaluated by the official test cases to determine your final grade after the deadline.