CPS721: Assignment 4

Due: November 11, 9pm
Total Marks: 110 (worth 4% of course mark)
In groups of no more than 3

Late Policy: Late submissions are penalized by $3^n\%$ when they are n days late.

Clarifications and Questions: Please use the discussion forum on the D2L site to ask questions. These will be monitored regularly. A Frequently Asked Questions (FAQ) page will also be created. Only ask questions by email if it is specific to your submission. Any questions asked with less than 12 hours left before the deadline are not likely to be answered.

PROLOG Instructions. Your code must run on ECLiPSe 7.1. If you cannot install this on your own machine, test it on the lab machines or the department moon servers.

You are NOT allowed to use ";" (disjunction), "!" (cut), "->" (if-then), `setof`, `bagof`, `findall`, or any other language features and library predicates that are not covered in class. Similarly, you should only use "=" and "is" for equality, and "not" for negation. Do NOT use "=:=", "\=", "+\", or other such commands. You will be penalized if you use these commands/features. You are only allowed to use ";" to get additional responses when interacting with PROLOG from the command line (this is equivalent to using the "More" button in the ECLiPSe GUI).

You ARE allowed to use the built-in `member` or `append` methods if they are useful to you.

Academic Integrity and Collaboration Policy: Any submission you make must be the work of your group and your group alone. You are also not allowed to post course materials (including assignments, slides, etc.) anywhere on the web, and you are also prohibited from using generative AI systems or online "help" websites for completing assignments.

Submission Instructions: You should submit ONE zip file called `assignment4.zip` on D2L. It should contain the following files:

```
assignment4_submission_info.txt
movie_kb.pl
movie_q_and_a_system.pl
q4_queries.txt
q5_parser.pl
```

All these files have been given to you and you should fill them out using the format described. Ensure your names appear in all files — especially `assignment4_submission_info.txt` — and your answers appear in the file sections as requested. Do NOT edit, delete, or add any lines that begin with "`%%%%% SECTION:`". These are used by the auto-grader to locate the relevant code. You will lose marks if you edit such lines.

Your submission should not include any files other than those above. You will be penalized for using compression formats other than `.zip`. Submissions by email will not be accepted.

You may submit your solution multiple times. Doing so will simply overwrite your previous submission. We will always mark the last submission your group makes.

1 Building a Database [10 marks]

This assignment will exercise what you have learned about natural language understanding and processing. To do so, you will be developing a question and answering system for a movie database. Ultimately, we would like to be able to interact with the system with statements like "what movie from 2022 was directed by Steven Spielberg?" For this assignment, we will restrict the system to only handle noun phrases, just as we did in class.

To build such a system, you will need a database of facts, a lexicon, and a parser. In this question, you will build this database. To do so, add atomic propositions to the file `movie_kb.pl` in the section called `database`. The atomic propositions should ONLY use the following predicates:

- `releaseInfo(MovieName, Year, Length)` - a movie named `MovieName` was released in year `Year` and is `Length` minutes long.
- `directedBy(MovieName, Director)` - the movie named `MovieName` was directed by `DirectorName`.
- `actedIn(Name, Movie, CharacterName)` - an actor with name `Name` plays a character named `CharacterName` in the move named `Movie`.
- `movieGenre(Movie, Genre)` - the movie with name `Movie` belongs to the genre `Genre` (*ie.* `comedy`, `drama`, `horror`, etc.).

Notice that these are the same predicates as in assignment 1, with the addition of `movieGenre`.

Your knowledge base should contain at least 10 movies. Each should have exactly one `releaseInfo` statement. Each movie should also have at least one genre, at least one director, and at least two actors (though movies can share actors). Note, this means that movies may have may have multiple directors, multiple genres, and multiple actors. An actor may also play multiple characters in any single movie.

In addition, consider the following:

- You may reuse parts of your Assignment 1 knowledge base.
- You will use this knowledge base for testing, so we suggest you read the rest of the assignment first to see what kind of statements will be useful to fully test your system.
- As on Assignment 1, use constants for your movie names, actor names, and director names. For names or movie titles, use underscore as blanks. For simplicity, you should also not include any punctuation or accents in names as well.

2 Helpers [20 marks]

For this question, you will write predicates that will be useful for creating your system. All of these helpers, and any others you decide to include in your system should be put in section `lexicon_and_helpers` in `movie_q_and_a_system.pl`. Note that for all these helpers, your predicates should work when the arguments are given as variables. We will largely test then in that context.

a. [**8 marks**] Create rules for the predicate `movie(Name)`, which succeeds if `Name` is the name of any movie. If the query `movie(X)` is made where `X` is a variable, then you should be able to use "More" to enumerate all movies in your KB. Similarly, create rules for the predicates `actor(X)`, `director(X)`, `character(X)`, `genre(X)`, `releaseYear(X)`, and `movieLength(X)`, that are defined analogously.

b. [**3 marks**] Create a predicate `newDirector(Name)` which only succeeds if `Name` is the name of a "new" director. Here, we say a director is "new" if they directed a movie during the current year, but they never directed a movie before this year. However, the current year should not be hard-coded as 2025 in the rules defining this adjective. Instead, you should use the `currentYear(Y)` predicate to retrieve the current year. Taking this approach makes it much easier to update your system when the year changes. We have given you such an atomic statement in `movie_q_and_a_system.pl`, but we may test your solution using KBs where 2025 is not the current year.

c. [**3 marks**] Define `newActor(Name)`, meaning that `Name` is the name of an actor who acted in some movie from this year, but never in a previous year. This predicate is defined similarly to `newDirector` from part (b).

d. [**3 marks**] Define `genreDirector(Name, Genre)`, meaning that a director with the name `Name` has made at least two different movies that are of the given genre. This will be used below to define what is means to be a "comedy director" or a "horror director", which we are defining as meaning that the director has a "history" of making movies in that genre. Importantly, this predicate should fail if the director has only made a single movie in that genre.

e. [**3 marks**] Define `genreActor(Name, Genre)`, meaning that an actor with the name `Name` has acted in at least two different movies that are of the given genre. This will be used below to define what is means to be a "comedy actor" or "horror actor", which we are defining as meaning that the actor has a history of being in movies in that genre. Importantly, this predicate should fail if the actor has been in only a single movie in that genre.

3 Building a Lexicon [50 marks]

You will now build a lexicon of articles, common nouns, proper nouns, adjectives, and prepositions. Each should be added to the `lexicon_and_helpers` section of `movie_q_and_a_system.pl`. Please ensure you have completed Q1 and Q2 before doing this part.

Below, we describe what your lexicon must include. Feel free to add additional words if you would like to extend the functionality of your system, though you will not get marks for doing so.

IMPORTANT: Your rules for defining what satisfies the different terms in the lexicon should not be "hardcoded" based on your KB. This is because we will test your rules with KBs other than your own. The helpers in the Q2 will be helpful for this purpose.

In addition, we will test your lexicon using the default parser given in `original_parser.pl`. You should NOT edit this file. In particular, we will read in our own KB, your lexicon, your helper predicates, and the default parser for testing your lexicon.

For convenience, we have added a version of the `what` predicate that takes in a string, tokenizes that string, converts those tokens to constants, puts those constants in a list, and then calls the "usual" `what` predicate. This means that both of the following are equivalent:

- `what([a, bong_joon_ho, film, released_in, 2023], X).`
- `what("a bong_joon_ho film released_in 2023", X).`

You do not need to edit the code that does this, or even understand it. It is just there to make testing your code a bit easier. You are free to use either version in your tests.

Article

Your lexicon should include the articles `a`, `an`, `the`, and `any`. In this assignment, the meaning of these are all equivalent. For example, both `[a, movie, ...]` and `[an, movie, ...]` are valid.

We recommend implementing this part of the lexicon first.

Common Nouns

Your lexicon should include the common nouns `movie`, `film`, `actor`, `director`, `character`, `length`, `running_time`, `genre`, and `release_year`. Note the following

- The words `movie` and `film` can be used interchangeably. The same is true of `length` and `running_time`.
- Notice that we are not distinguishing between an "actor" and an "actress" in this system.

Note, once you have added some articles and common nouns, you can start testing your system with queries such as `what([a, movie], X)` or `what([a, running_time])`, which should allow you to iterate over all movies, or all unique running times in your database, by calling "More".

Adjectives

Your lexicon should define the following adjectives:

- `three_hour` - a movie is a three hour movie if it is 3 hours or longer.
- `short` - a movie is a short movie if it is less than 60 minutes in length
- `new` (movie) - a movie is new if it was released in the current year. However, the current year should not be hard-coded as 2025 in your rules defining this adjective. Instead, you should use the `currentYear(Y)` predicate to retrieve the current year from the KB. Taking this approach makes it much easier to keep your system up to date, but still working (like when the year changes). We have given you such a predicate, but we may test your solution using KBs where 2025 is not the current year.

- `new` (actor or director) - an actor is "new" if they never acted in any movie prior to the current year. Similarly, a director is "new" if they never directed any movie before the current year.
- Each genre can be used as an adjective for a movie. For example, `[a, horror, movie]` is a movie that has the genre `horror` (note, we will still count it as a horror movie, even if it has multiple genres, as long as one is `horror`). Your solution should be able to handle any arbitrary genre in the knowledge base, not just the one you use for testing.
- Each genre can be used as an adjective for an actor or director, as long as they have a history of appearing in movies of that type (meaning, at least two movies of that genre). For example, a `[a, comedy, director]` should only match on someone who has made at least two different comedy movies.
- Actor and director name can be used as the adjective of a movie. For example, `[a, clint_eastwood, movie]` should match on any movie in your KB that is either directed by Clint Eastwood, or that Clint Eastwood acted in.

Once you have added all the articles, common nouns, and adjectives listed above, you should be able to handle queries similar to the following:

- `what([a, scifi, film], M).`
- `what([a, three_hour, horror, movie], M).`
- `what([a, new, steven_spielberg, movie], M).`
- `what([any, short, denzel_washington, movie], M).`

Proper Nouns

Your lexicon should include all movie names, actor names, director names, and character names. Again, recall that we will be testing your system using our own knowledge base, so your rules should be able to handle having the knowledge base changed to have other names.

In addition to the above, any number (as used for release year or movie length) is also considered a proper noun (this is because you would say "a movie released in 2020" instead of "a movie released in the 2020"). You may find it useful to use the library predicate `number(X)` , which is true if `X` is a number and false otherwise, for this purpose.

Note that proper nouns are a referent to themselves and thus, you can handle them with rules of the form:

```
proper_noun(X) :- person(X).    % X is the name of the person in this context
```

Prepositions

Your lexicon should define the following prepositions:

- `by` - to refer to a movie directed "by" some director.
- `with` - to refer to a movie "with" a given actor (*ie.* "a movie with Tom Cruise"). "with" can also be used to refer to a movie "with" the given character (*ie.* "the film with Lex Luthor").
- `in` - to refer to an actor in a given movie (ie. "the actor in Jaws"), or the character in a given movie (*ie.* "the character played by Roy Scheider in Jaws").
- `from` - to refer to a movie from a given year (*ie.* "a horror movie from 2015").
- `released_in` - similar to "from" in that it refers to the year that a movie was **released in**.
- `played_by` - indicates that a given character was played by the actor in question.

- `of` - will be used in two contexts. The first is phrases of the form "The release year of Jaws". The second is of the form "The length of Jaws".

If you have all parts of the lexicon and KB completed, you should be able to successfully do queries like the following:

- `what([a, martin_scorscese, film, with, leonardo_dicaprio], M).`
- `what([the, genre, of, the, pt_anderson, movie, released_in, 2025], M).`
- `what([the, length, of, sinners], L).`
- `what([the, release_year, of, the, documentary, movie, by, sarah_polley], Y).`
- `what([the, character, in, glass_onion, played_by, janelle_monae], C).`
- `what([an, actor, in, an, action, movie, released_in, 1980], A).`

This is just a sample of the types of queries we may test with.

4 Testing your Lexicon [10 marks]

For this question, you will test your lexicon and KB by trying queries using the `what` predicate and the simple noun phrase parser given in class. This parser has already been added to `original_parser.pl`.

To that end, create 10 queries different from the examples given in this document (though they may be similar). Write these queries, and the log of the output of these queries in the file `q4_queries.txt`. For at least 2 of the queries, you should "More" to find all the solutions to your queries. Repeated answers are fine, and to be expected.

Please note the following:

- This part of your submission will not be evaluated on whether your systems succeeds or not on your queries, but whether they use only the lexicon given to make valid queries. Having the queries fail is thus ok.
- The purpose of this question is to get you to effectively test your system. Thus, you should think about what queries give you confidence you have defined the lexicon correctly.
- You are free to use the version of the `what` predicate that takes in a string in your tests.

5 Parser Features [20 marks]

For these questions, you will edit/augment the parser code to add extra features to your system. In particular, you will edit the parser in `q5_parser.pl` to add these features. We will test this code by importing your `lexicon_and_helpers` code in `movie_q_and_a_system.pl`, along with the current year, the definition of `what`, and the code in `q5_parser.pl`.

Since this part of the assignment depends strongly on having a working lexicon and KB, we strongly suggest you do not do this part of the assignment until you have completed all previous parts.

a. [**10 marks**] Add to your system the ability to handle prepositional phrases of the form "between X and Y". For example, your system should be able to handle the following:

- `what([an, movie, with, a, release_year, between, 2020, and, 2022], M).`
- `what([a, film, with, a, length, between, the, length, of, fury_road, and, 300], S).`

Note that "between" is not inclusive. Thus, for the first example above, only movies with a release year of 2021 are applicable.

Most of the marks will be given for handling input values that are found in the knowledge base. However, for full marks, your parser will need to handle queries that mention an integer never mentioned in the KB.

b. [**10 marks**] Add to your system the adjective `oldest` which can be used as follows:

- `what([the, oldest, movie], M).`
- `what([the, oldest, action, movie, with, charlize_theron], M).`

You can assume that when there is a sequence of adjectives, `oldest` must be the first one. For example, we can have phrases like `[the, oldest, action, movie]` but we cannot have phrases of the form `[the, action, oldest, movie]`.