MSc Artificial Intelligence Programming in Prolog, 2004

Second Assessed Assignment:

Six Degrees of Kevin Bacon

Tim Smith (tim.smith@ed.ac.uk)

November 8th, 2004

1 Introduction

This is the second assignment (of two) for the 2004 MSc Artificial Intelligence Programming in Prolog module. It is worth 20% of the total mark for the module. It should take 15-20 hours to complete (on average).

The assignment should be submitted by 4pm on Friday 12th December (week 11) (notice the change of deadline from that noted in the coursenotes). Submission will be via the 'submit' command on the Informatics DICE network. Details of this are at the end of the assignment.

2 Task Summary

Your task will be to implement a text-based interface for a movie database. The database contains facts representing actors (male only) and the films they have appeared in. This information is represented as Prolog facts and the whole database can be consulted by SICStus. This allows the user to query the database directly using normal Prolog queries. However, this means that the database can only be accessed by users who understand Prolog. A better interface would allow the user to type natural language queries which will be converted into the correct Prolog queries and used to search the database for answers. The answers would then be returned as English sentences. It is your task to implement such an interface.

An example interaction could be:

```
| ?- go.
|: 'how can kevin_bacon be connected to sean_connery?'

sean_connery was in the_first_great_train_robbery with
donald_sutherland_(i) in 1979. donald_sutherland_(i) was in
animal_house with kevin_bacon in 1978.

yes
```

Your program will have to deal with two different types of query:

- direct queries of facts in the database, and
- queries about connections between actors.

The final set of queries will allow the user to generate solutions to the 'Six Degrees of Kevin Bacon' game: a party game in which the task is to connect one actor to another using the films they have appeared in. If the two actors A and B have appeared in the same film then they are said to have a separation of 1. If A and B have not appeared in the same film then another actor, C, is found who has appeared in a film with A and a film with B. A and B are then said to have a separation of 2 (A-C-B). If one extra actor is not sufficient to link the two target actors then more actors can be added until a connection is found. In practice, most actors can be connected by less than three films (average degree of separation is 2.95). The inventors of this game seemed to think Kevin Bacon was particularly well connected so they blessed him with the honour of having the game named after him (that and the fact that "Kevin Bacon" sounds a bit like "separation").

3 The Prolog Movie Database (PMDB)

The movie database used for this assignment was translated from the source files for Internet Movie Database (www.imdb.com). Our database contains all male actors who appeared in more than one film since 1958 (the year Kevin Bacon was born). The database entries are ordered according to actor surname and film title.

An example entry is:

```
actor('kevin_bacon', 'animal_house', 1978, 19).
```

Each entry consists of:

Argument 1 (ACTOR) = the actors full name as a string (enclosed in single-quotes);
Argument 2 (FILM) = the full name of the film as a string;
Argument 3 (DATE) = the year of the film's release as a number;
Argument 4 (RANK) = the actors rank in the film according to the credits (e.g. were they the star, 1, or uncredited, 0).

The ACTOR and FILM data objects are stored as lower-case Prolog atoms that contain no syntax or spaces. Queries to the database must match the format of each data object exactly for unification to occur.

The database is available for download from the course website. There are three versions of the database. The versions should be used for different sections of the assignment:

- **`actors_1deg.pl`**
    - This is a small file containing only the actors who can be connected to Kevin Bacon via one film. This should be consulted for the first section of the assignment, Querying the Database.

- **`actors_2deg.pl`**
    - This is a larger file (7.8mb) and should only be consulted when playing the 'Six degrees of Kevin Bacon' game in the final section of the assignment. It contains all actors who can be connected to Kevin Bacon through two or less films.
- **`actors_Ndeg.pl`**
    - This is a huge file (37.7mb) that should only be used for the final sample queries of the 'Six degrees of Kevin Bacon' game. It contains all actors who have appeared in more than one film since 1958. It will take a few minutes for Sicstus to consult this file so it is not very practical to be consulted during development of your code.

These database files must not be altered either by hand or by your Prolog program (via append, retract, etc). They should be consulted by your program and queried whilst in Sicstus' memory buffer. This is the most efficient way to access the database.

4 Outline of Tasks

To develop a text-based input for the PMDB you will need to accommodate two different sets of queries. The first set of queries address the facts in the database directly whilst the second set plays 'Six Degrees of Kevin Bacon' with the database. Both sets of queries should be accepted by a single interface program.

Your program should be activated by the command:

```
|?- go.
```

It should then prompt the user for their question. The question should be input as a string (enclosed in single-quotes) and terminated with the full-stop. All letters should be lower-case and the names of actors and films should be written as atoms, just as they appear in the database.

```
|:'was kevin_bacon in sleepers?'.

yes.
```

Appropriate answers should be displayed to the user. If the answer is more than yes/no it should be written as a sentence (unless otherwise instructed; see example output in the next section). The user should then be asked for another question.

```
Would you like to ask another question?

|:no.

I hope I have been of help.
```

5 Querying the Database

The first set of queries your program must accept questions the database directly. The user should be able to type English sentences asking about the data stored in the database and receive clear and informative answers. Below are sample questions your program **must** accept and the expected answers.

(n.b. facts are stored in the database in this form: actor(ACTOR, MOVIE, DATE, RANK).)

1. Q: 'was kevin_bacon in sleepers?'.
   A: yes.
2. Q: 'who was in sleepers?'.
   A: kevin_bacon 1

   billy_crudup 2

   robert_de_niro 3

   ron_eldard 4

   vittorio_gassman 6

   dustin_hoffman 7

   terry_kinney 8

   bruno_kirby 9……..
   were in Sleepers, 1996.

   (Your program should list all actors in RANK order and their RANK written next to them)

3. Q: 'which films has kevin_bacon appeared in?'.
   A: kevin_bacon has appeared in

   animal_house 1978
   starting_over 1979
   friday_the_13th 1980
   hero_at_large 1980
   only_when_i_laugh 1981
   diner 1982
   forty_deuce 1982
   enormous_changes_at_the_last_minute 1983
   footloose 1984
   quicksilver 1986
   trains_and_automobiles_planes 1987
   white_water_summer 1987
   criminal_law 1988…….

   (*all Kevin Bacon's films should be listed and ordered by DATE)

4. Q: 'which year was sleepers made?'.

A: sleepers was made in 1996.

5. Q: 'which films did kevin_bacon make in 2000?'.
   A: kevin_bacon appeared in hollow_man, my_dog_skip, and we_married_margo in 2000.
6. Q: 'did kevin_bacon star in the_woodsman?'.
   A: yes.

(* we will interpret an actor as 'starring' in a film when they have a RANK of 1)

7. Q: 'which films has kevin_bacon starred in?'.
   A: kevin_bacon has starred in
   the_woodsman 2004
   white_water_summer 1987
   tremors 1990
   telling_lies_in_america 1997
   stir_of_echoes 1999
   sleepers 1996
   shes_having_a_baby 1988
   quicksilver 1986
   queens_logic 1991
   pyrates 1991
   he_said_she_said 1991
   footloose 1984
   the_big_picture 1989
   balto 1995
   the_air_up_there 1994

Your program **must** accept these queries but it should also be able to understand the same questions asked in different ways e.g. "was kevin_bacon in sleepers?" could be re-written as "did kevin_bacon appear in sleepers?" . The greater the range of questions your program recognises the more marks you will be awarded for this section.

5b Building an interface

To be able to respond to these queries your program must:

1. **Prompt the user for a question**. The prompt should also instruct the user on what form the question should be in. Remember that this has to match the format of the database.

2. **Tokenize the sentence**. Reading a string as input from the user prompt your program should convert that string into a list of words. You should use `name/2` to convert the string into a list of ASCII codes and then use the syntax of the sentence to identify word boundaries. Don't forget to identify punctuation as this might be important to the meaning of the sentence.

3. **Pass the wordlist to a DCG**. Once you have identified the words, you need to extract the meaning of the sentence. This requires you to write a DCG with enough rules and a large enough lexicon that it can handle all the specified queries and their variations.
    - The DCG does not have to be 100% grammatically correct; this is a Prolog assignment not a computational linguistics assignment. You should write the DCG so that it can handle the particular sentence structures the user will be using. It does not need to be able to identify grammatical structures that bare no relation to acceptable questions.
    - The terminals of your DCG will be the actors and films of the database. If these cannot be identified appropriate responses should be passed back to the user.
      ```
      |:'was donald_sutherland in animal_house?'
      I am sorry, I do not know the actor donald_sutherland
      ```
      (* the database entry is donald_sutherland_(i))

    - Similarly, if the whole sentence cannot be parsed then an informative response should be given to the user. A general response stating that the query was not understood is sufficient but more specific responses will be rewarded with more marks.

    - During parsing the DCG should also construct a Prolog representation of the query that needs to be called by Prolog to answer the question. This must encapsulate the meaning of the sentence and be in a form that it can be called as a Prolog command (i.e. a compound structure with functor and arguments).
      The question:
      ```
      'was kevin_bacon in sleepers?'
      ```
      should generate the query:
      ```
      actor(kevin_bacon,sleepers,_,_).
      ```
      which can be either be called (using call/1) or just placed as a command in your program.

    - The meaning of a query might require multiple Prolog commands to represent it. These can be placed in a list in the head of a DCG rule and called in order.

    - Your DCG should re-use structures as much as possible. For example, if one query contains a prepositional phrase (PP) then the same grammar rules used to parse that PP should be used to parse other queries containing PPs. You DCG will be assessed both on its completeness and its economy of code (the DCG should be small but intelligently structured).

4. **Formulate an answer to the query**. The meaning of the query should be represented in a way that it can generate any computation needed to find an answer. This might be a simple test of a fact in the database or it might require more complex recursive processing. Either way, your DCG should generate the appropriate prolog commands which can be carried out by your program.

- **Hint**: define and test any extra predicates outside of your DCG before trying to integrate them.

5. **Generate an appropriate response.** Once the Prolog commands have been tested the results should be gathered and an appropriate response presented to the user. This response should be based on the example responses shown in the last section. You do not need to write a DCG to produce this output as simple patterns of `write/1` commands should suffice.

6 Six Degrees of Kevin Bacon

The second part of the assignment requires you to adapt you program so that it can find solutions to the 'Six Degrees of Kevin Bacon' game. The user should be able to pose questions about the connections between actors. Your program should accept the following queries:

1 degree of separation

1. Q: 'can kevin_bacon be connected to dustin_hoffman?'.
   A: yes.

2. Q: 'how can kevin_bacon be connected to dustin_hoffman?'.
   A: 'dustin_hoffman was in sleepers with kevin_bacon in 1996.'

2 or more degrees of separation

1. Q: 'can kevin_bacon be connected to sean_connery?'
   A: yes.

2. Q: 'how can kevin_bacon be connected to sean_connery?'
   A: sean_connery was in the_first_great_train_robbery with donald_sutherland_(i) in 1979. donald_sutherland_(i) was in animal_house with kevin_bacon in 1978.

3. Q: 'what degree of separation is sean_connery from kevin_bacon?'.
   A: 'sean_connery is separated from kevin_bacon by 2 degrees.'

The first set of queries can be answered using the **actors_1deg.pl** database. The second set require either **actors_2deg.pl** or **actors_Ndeg.pl.** You are advised to use actors_2deg.pl for most of your queries as it is quicker to consult and search compared to actors_Ndeg.pl . However your final program will be tested for degrees of separation up to 3. For example:

|: 'how can kevin_bacon be connected to thomas_aagren?'

thomas_aagren was in pusher with thor_nielsen in 1996,

thor_nielsen was in an_enemy_of_the_people with kenneth_white_(i) in 1978, and

kenneth_white_(i) was in Apollo_13 with kevin_bacon in 1995.

You will need to implement a search algorithm for generating these connections between actors. The solution generated should always be the shortest route between two actors. A solution with 3 degrees of separation should not be generated if a 2 degrees solution exists. The easiest way to ensure that the shortest route is always found first is to perform an **iterative deepening search**. This exhausts all links at one depth before moving to the next depth. Given the size of our database, it is also the most efficient search strategy. You will find an example iterative deepening search algorithm in the lecture notes.

To complete this section you will need to:

1. Write an iterative deepening algorithm that takes two actors and generates a list of movies and actors that connect them.

2. Write a predicate that can write out the path between the two actors in the form shown in the example queries above.

3. Adapt the DCG from the previous section so that it can accept queries of the form: `'how can kevin_bacon be connected to thomas_aagren?'`. The DCG should parse the query, identify the actors, and formulate the appropriate prolog command necessary to find an answer (this should be the search predicate you just defined).

4. If the query cannot be understood or a connection not found between actors (this may happen when trying to connect actors with a degree of separation of 3 whilst consulting actors_2deg.pl) your program should generate clear and informative responses.

Your program should always return an answer if one exists. As the degree of separation of the actors increases and the size of the file database being searched increases so will the search time. However, all queries listed above should generate a response within a couple of minutes. If your program is taking a long time to find solutions to these queries then it is probably doing something wrong.

Extra Bonus Queries: 5% of the assignment mark will be reserved for inventive variations of 'Six degrees of Kevin Bacon' search problem. Your extra queries should perform a variation of the 'six degrees' search problem that is not already listed above. These extra queries will be marked for originality, appropriate use of a search strategy, and the range and quality of input and output sentences. An example variation could be: `'can kevin_bacon be connected to sean_connery via bagpuss?'`

7 Assessment

The assignment will be marked out of 100%:

- 20% for range and variety of all queries accepted by the DCG;
- 10% for appropriate handling of erroneous or unidentifiable queries;
- 15% for correct responses to section 1 queries ('Querying the database');
- 20% for the implementation of the iterative deepening search algorithm;
- 15% for the accuracy of the 'Six degrees of Kevin Bacon' queries;
- 5% for the bonus 'Six Degrees of Kevin Bacon' queries;
- 5% for clarity and presentation of all answers;
- The remaining 10% will be given for general programming style, such as:
    - Clear layout of Prolog code,
    - Clear and useful commenting (*if I don't know what your code does I can't mark it! It is in your interest to comment your code fully),
    - Efficient and concise predicate definitions.

8 Hints and Advice

- You can use the lists module provided in SICStus to save you having to define predicates like member/2 and append/3 again. Modules are compiled instead of consulted so they run much faster but the downside is that you can't trace them. Write **:- use_module(library(lists)).** at the top of your file and the module will load automatically. Details of what this module contains can be found in the Sicstus manual: **http://www.sics.se/sicstus/docs/latest/html/sicstus.html/Lists.html#Lists**

- Read this specification carefully, several times, before thinking about starting on the exercise.

- Don't be intimidated by the length of this specification. Good Prolog programs are usually quite short, and a good solution to this exercise will almost certainly be shorter than you expected. If you find yourself writing what seems like a lot of Prolog, you might be making things more complicated than they need to be. Take a step back and rethink.

- Take lots of care with laying out your code clearly. Use lots of white space, and plenty of clear comments. Part of your mark is for formatting and commenting of code. If you don't take care over this, you will lose easy marks unnecessarily.

- If you're having problems, remember that you can use trace/0 and notrace/0 to enable Prolog's tracing, and spy/1 to look at individual predicates. You can also use write/1 to write 'flags' that tell you where your code has got to.

9 Plagiarism

This is an **individual** exercise, so what you submit should be your own work. Please make sure that you understand the Division's Guidelines on Plagiarism, which you can find here: http://www.informatics.ed.ac.uk/admin/ITO/DivisionalGuidelinesPlagiarism.html

If you submit work that isn't wholly your own, you must say so clearly in your submission. Discussing general issues concerning an exercise with other students is a good thing, and is expected, but genuine collaboration on all or part of an assessed exercise must be explicitly acknowledged and will be penalised.

10 How to submit

Your submission for this exercise should take the form of a single Prolog file containing all the predicates needed for the example queries to be answered. Full comments should be included so that the logic of your program can be understood.

- Put your name and matriculation number at the top of the file. Make sure that they're commented out, so that they don't stop the file loading into SICStus.
- Make sure that your submission loads into the version of SICStus installed on the DICE network. Make sure that comments are properly commented. If your submission fails to load, and there are simple and obvious changes which allow it to load (like commenting out something which should have been commented already), I might make those changes, but you will lose marks because of that. If the file needs more significant corrections inorder to load, I won't attempt any changes, and you will be marked down accordingly. If you make any changes to your submission at the last moment, **no matter how small they seem**, test that the file loads before you submit.

Submit the file electronically, by 4pm on Friday, 12th December (the end of week 11), on any DICE machine using the command:

```
submit msc aipp 2 <filename>
```

Late submissions will not be accepted without a extremely good reason. If you have a case for an extension, please let me know as soon as possible.