Department of Computer Science
University of Pretoria

Programming Languages
COS 333

Practical 3: Logic Programming

September 7, 2022

1 Objectives

This practical aims to achieve the following general learning objectives:

- To gain and consolidate some experience writing logic programs in Prolog;
- To consolidate the concepts covered in Chapter 16 of the prescribed textbook.

2 Plagiarism Policy

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to `http://www.ais.up.ac.za/plagiarism/index.htm` (from the main page of the University of Pretoria site, follow the `Library` quick link, and then click the `Plagiarism` link). If you have any form of question regarding this, please consult the lecturer, to avoid any misunderstanding. Also note that the principle of code re-use does not mean that you should copy and adapt code to suit your solution. Note that all assignments submitted for this module implicitly agree to this plagiarism policy, and declare that the submitted work is the student's own work. Assignments will be submitted to a variety of plagiarism checks. Any typed assignment may be checked using the Turnitin system. After plagiarism checking, assignments will not be permanently stored on the Turnitin database.

3 Submission Instructions

Upload your practical-related source code file to the appropriate assignment upload slot on the ClickUP course page. You will be implementing all the practical's tasks in the same file. Name this file `s99999999.pl`, where `99999999` is your student number. Multiple uploads are allowed, but only the last one will be marked. The submission deadline is **Monday, 12 September 2022, at 23:00**.

4 Background Information

For this practical, you will be writing programs in SWI-Prolog version 7.6.4:

- Write all your Prolog programs (consisting of facts and rules) in a single ASCII text file. The SWI-Prolog interpreter is usually launched using the `prolog` or `swipl` command. More information on running the interpreter is provided in the relevant `man` page. Use the `-l` switch to load your source file. This will load your program source file and then launch the Prolog interpreter's interactive mode, from which you can type queries to which Prolog will provide feedback. As was the case for the Guile Scheme interpreter, only Prolog predicate implementations are provided in the ASCII source file. You should not provide queries in your program source code, but instead type these queries in the interpreter's interactive mode.
- In interactive mode, a `true` response means that the interpreter can prove your query to be true, while a `false` response means that the interpreter cannot prove your query to be true. If you include variables in your query, the interpreter will respond with a value that makes the query true, or `false` if it cannot find such a value. Pressing `r` after a query response will re-query the interpreter, while pressing `Enter` will end the query. Pressing `Ctrl+d` will exit the interpreter.
- The course ClickUP page contains documentation related to SWI-Prolog [1], which contains detailed information on the operation of the SWI-Prolog interpreter, and details on the implementation of the Prolog programming language that SWI-Prolog provides.
- **Note that you may only use the simple constants, variables, list manipulation methods, and built-in predicates discussed in the textbook and slides. You may NOT use any more complex predicates provided by the Prolog system itself. In other words, you must write all your own propositions. Failure to observe this rule will result in all marks for a task being forfeited.**
- **You may implement and use the propositions defined in the textbook and slides (e.g. `member`, `append`, and `reverse`). Note that you must provide the implementation for any of these propositions in your source file. Also note that there are some built-in propositions that correspond to the propositions defined in the textbook, which you may not use.**
- You may implement helper propositions if you find them necessary.

5 Practical Tasks

For this practical, you will need to explore and implement logic programming concepts, including list processing. All of the following tasks should be implemented in a single source code file.

5.1 Task 1

Write a Prolog program containing facts of the following form:

```
father(bill, jake).
father(bill, shelley).
father(jake, ted).
father(ron, liz).
mother(mary, jake).
mother(mary, shelley).
mother(janet, ted).
mother(shelley, liz).
```

Do not provide any additional facts other than `father` and `mother` propositions. **Failure to do so will result in a zero mark for this task**.

You must now implement a `cousin` proposition. A person's cousin is the son or daughter of their parent's sibling. In the above example, `ted` and `liz` are cousins, because `jake` and `shelley` are siblings, and `ted` is the son of `jake`, while `liz` is the daughter of `shelley`. Also ensure that it is not possible for someone to be considered their own cousin.

Hint: To test whether invalid objects are matched by your proposition, try entering queries involving variables, such as `cousin(X, Y)`. This will list all objects for `X` and `Y` that satisfy the proposition.

5.2 Task 2

Write a Prolog proposition named `countOccurrences` that has three parameters. The first parameter is a constant (i.e. a symbolic atom or an integer), and the second and third parameters are simple lists (i.e. lists containing only constants). The proposition defines the third parameter to be the number of occurrences of the first parameter in the second parameter. To illustrate the use of the `countOccurrences` proposition, consider the following queries and responses:

```
?- countOccurrences(a, [], X).
X = 0.

?- countOccurrences(a, [c, d, f], X).
X = 0.

?- countOccurrences(a, [c, a, c, a, a, b], X).
X = 3.
```

Note that the query `countOccurrences(a, [c, d, f], X)` produces a count of 0 because `[c, d, f]` does not contain the symbolic atom `a`.

Test the `countOccurrences` proposition using the provided example code, as well as your own test input, and verify that the proposition works as you expect.

Hint: Refer to the slides and textbook for details on the built-in `not` predicate. This may be useful for handling cases where two constants do not match one another.

6 Marking

Each of the tasks will count 5 marks for a total of 10 marks. Both the implementation and the correct execution of the programs will be taken into account. Your program code will be marked offline by the teaching assistants. Make sure that you upload your complete program code to the appropriate assignment upload slot, and include comments at the top of your program source file, explaining how your program should be executed, as well as any special requirements that your program has. Do not upload any additional files other than your source code.

References

[1] Jan Wielemaker. *SWI-Prolog Reference Manual.* University of Amsterdam, January 2018.