Department of Computer Science
University of Pretoria

Programming Languages
COS 333

Practical 3: Logic Programming

August 29, 2023 (Updated September 7, 2023)

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

Plagiarism is a serious form of academic misconduct. It involves both appropriating someone else's work and passing it off as one's own work afterwards. Thus, you commit plagiarism when you present someone else's written or creative work (words, images, ideas, opinions, discoveries, artwork, music, recordings, computer-generated work, etc.) as your own. Note that using material produced in whole or part by an AI-based tool (such as ChatGPT) also constitutes plagiarism. Only hand in your own original work. Indicate precisely and accurately when you have used information provided by someone else. Referencing must be done in accordance with a recognised system. Indicate whether you have downloaded information from the Internet. For more details, visit the library's website: `http://www.library.up.ac.za/plagiarism/`.

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 **Tuesday, 26 September 2023, at 12:00**.

4 Background Information

For this practical, you will be writing programs in SWI-Prolog version 9.0.4:

- Write all your Prolog programs (consisting of facts and rules) in a single source file. To do this, launch the SWI-Prolog interpreter in Windows, and select "New..." under the "File" menu. Type a file name, and click "Save". An edit dialogue will then pop up, through which you can write your program source file. Source code files are saved by selecting "Save buffer" under the "'File' menu. Note that SWI-Prolog is sensitive to end of line characters, so it is strongly recommended that you use this built in editor to write your programs. Also, be sure to place facts and rules on separate lines. Note that only Prolog predicate implementations are provided in your source file. You should not provide queries in your program source code.
- In order to test your implementation, select "Consult..." under the "File" menu, then choose the program source file you have written. You can then type in queries in the main SWI-Prolog window, which is in 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.
- 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

Define a series of facts relating to languages spoken by individuals, such as the following:

```
speaks(catalina, spanish).
speaks(pablo, spanish).
speaks(john, english).
speaks(rachel, english).
speaks(john, german).
speaks(johanna, german).
```

Note that the **only** facts you are allowed to define must use the `speaks` proposition. If you define any facts using additional propositions, you will **forfeit all marks for this task**. The `speaks(X, Y)` proposition means that person `X` speaks language `Y`. For simplicity, assume that two languages can have a maximum of one person speaking both languages, and that a person can speak a maximum of two languages.

You must define the following propositions by means of rules that use the fact propositions listed above:

- The proposition `bilingual(X, L1, L2)`, which is true when person `X` speaks both languages `L1` and `L2`. Given the facts in the example above, `john` is bilingual because he speaks `english` and `german`. Ensure that the `bilingual` proposition does not identify a person as being bilingual where `L1` and `L2` are the same language.
- The proposition `canCommunicateTranslated(X, Y)`, which is true when person `X` and person `Y` speak different languages, but they have a person with whom they can both directly communicate who will translate for them (in other words, `X` has a person they can directly communicate with, who can also directly communicate with `Y`). Given the facts in the example above, `rachel` and `johanna` can communicate via a translator, because `rachel` speaks `english` and `johanna` speaks `german`, but they can both directly communicate with `john`, who can translate for them. Ensure that `X` and `Y` speak different languages.

Hint 1: It will make your implementation much easier if you use the `bilingual` propositions in your rule defining the `canCommunicateTranslated` proposition. Be sure to test both your propositions thoroughly, including both propositions that should be true as well as false.

Hint 2: To test whether invalid objects are included in your proposition, try entering queries involving variables, such as `canCommunicateTranslated(X, Y)`. If you re-query repeatedly, this will list all objects for `X` and `Y` that satisfy the proposition.

5.2 Task 2

Write a Prolog proposition named `maximumElement` that has two parameters. The first parameter is an integer, and the second parameter is a simple numeric list (i.e. a list containing only non-negative integers). The proposition defines the first parameter to be the maximum value contained in the second parameter. The maximum value in an empty list is `0`. To illustrate the use of the `maximumElement` proposition, consider the following queries and responses:

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

?- maximumElement(X, [1, 5, 2]).
X = 5.
```

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

Hint: You can use comparison operators as terms in Prolog. These operators are similar to what you are used to in imperative languages. The operators `<` (less than), `=<` (less than or equal to), `>` (greater than), and `>=` (greater than or equal to) are all valid. As a very simple example, you could use a comparison operator as follows:

```
lessThan(X, Y) :- X < Y.
```

The following queries are then possible:

```
?- lessThan(1, 2).
true.

?- lessThan(2, 1).
false.
```

6 Marking

Each of the tasks will count 5 marks for a total of 10 marks. Submit both tasks implemented in the same source code file. Do not upload any additional files other than your source code. Both the implementation and the correct execution of the propositions will be taken into account. **You will receive zero for a task that uses a language feature you are not allowed to use**. Your program code will be assessed during the practical session in the week of **Monday, 25 September 2023**.

7 Changes Since Last Version

Clarified that parameter list in Task 2 can be assumed to not contain negative values.

References

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