Department of Computer Science
University of Pretoria

Programming Languages
COS 333

Practical 4: Functional and Logic Programming

September 21, 2022

1 Objectives

This practical aims to achieve the following general learning objectives:

- To consolidate the concepts covered in Chapter 15 of the prescribed textbook.
- 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 files to the appropriate assignment upload slots on the ClickUP course page. Upload only the source code files for the Scheme and Prolog implementations. Name the Scheme implementation file `s99999999.scm`, and name the Prolog implementation file `s99999999.pl`. In both cases, `99999999` is your student number. Multiple uploads are allowed, but only the last one will be marked. The submission deadline is **Monday, 3 October 2022, at 23:00**.

4 Background Information

You will be implementing your first task in Scheme and your second task in Prolog. Please refer to the background information in practicals 2 and 3 for further information on how to write and interpret Scheme and Prolog programs. Guile version 2.2.3 will be used to assess your Scheme submission. SWI-Prolog version 7.6.4 will be used to assess your Prolog submission.

5 Practical Tasks

5.1 Task 1: Scheme

You must once again use the GNU Guile interpreter to run and test your submission for this task. For further information on the GNU Guile interpreter, please see the specification for Practical 2.

Once again, note that you may only use the following functions and language features in your programs, and that failure to observe this rule will result in all marks for this task being forfeited:

Function construction: `lambda`, `define`

Binding: `let`

Arithmetic: `+`, `-`, `*`, `/`, `abs`, `sqrt`, `remainder`, `min`, `max`

Boolean values: `#t`, `#f`

Equality predicates: `=`, `>`, `<`, `>=`, `<=`, `even?`, `odd?`, `zero?`, `negative?`, `eqv?`, `eq?`

Logical predicates: `and`, `or`, `not`

List predicates: `list?`, `null?`

Conditionals: `if`, `cond`, `else`

Quoting: `quote`, `'`

Evaluation: `eval`

List manipulation: `list`, `car`, `cdr`, `cons`

Input and output: `display`, `printf`, `newline`, `read`

Write a Scheme function named `last`, which is applied to one parameter, which is a simple list containing only atoms. The `last` function yields a list containing only the last element in the list provided as a parameter. The last element of an empty list is considered to be an empty list. Below are some examples of how the `last` function can be tested. The function application

```
(last '(a b c))
```

should yield the list `(c)`. As another example, the function application

```
(last '())
```

should yield an empty list. Note that you should test with a variety of different parameters, not only the ones provided above.

5.2 Task 2

You must once again use the SWI-Prolog interpreter to run and test your submission for this task. For further information on the SWI-Prolog interpreter, please see the specification for Practical 3.

Once again, take note of the following requirements for your submission:

- **Note that you may only use the simple constants, variables, list manipulation methods, and built-in predicates discussed in the textbook and slides, as well as the hint provided below. 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.**

Write a Prolog proposition named `sumPositiveListValues` that has two parameters. The first parameter is a list, and the second parameter is a numeric value. The proposition defines the second parameter to be the sum of only the positive values in the list. To illustrate the use of the `sumPositiveListValues` proposition, consider the following queries and responses:

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

?- sumPositiveListValues([10], X).
X = 10.

?- sumPositiveListValues([5, 2, 7], X).
X = 14.

?- sumPositiveListValues([-5, -2], X).
X = 0.

?- sumPositiveListValues([-2, 5, -5, 2, 1], X).
X = 8.
```

Test the `sumPositiveListValues` proposition using the provided example code, as well as your own test input, and verify that it works as expected.

Hint: Prolog provides standard numeric comparisons that can be used in rules. For example, the term `X < 0` will be true if `X` is negative. The following is a very simple example of how such a term can be used:

```
isNegative(X) :- X < 0.
```

In this example, the query `isNegative(4)` will be false, while the query `isNegative(-1)` will be true.

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 slots, and include comments at the top of your program source files, explaining how each program should be executed, as well as any special requirements that your programs have.