CSE-3401A
Functional and Logic Programming
Summer 2008
Report 2 Specification
Due: Wednesday, July 16, in class

July 3, 2008

Be sure to read www page on *On Academic Honesty* reachable from the home page for the course.

The files **functionals.lsp**, and **functionals-base.lsp** in the course directory on Prism contain the functionals and base support functions you need for this report. Unless explicitly told to your functions should not use explicit recursion. Unless explicitly told to do so do not use the functionals `sigma, iter, for, while`. For the Prolog exercise do not use material that occurs after and including Accumulators. In particular do not use cut (!) or not (\+). Unless indicated otherwise, all predicates should only give one solution. Do not have more than 6 tests for each function and predicate you write. To minimize your work avoid redundant tests; be sure that each test tests for something different. For example, inserting into a list away from the ends needs only one test, as, in general it makes no difference if, in a list of length 10, you insert at position 3,4,5,6,7, as all those are equivalent for the insertion algorithm.

To hand in

- The first page of the report you hand in is a standard cover page that you get from the course web pages.
- The body part of the report is a hardcopy of your answers in exercise order. For each exercise that does not require programming submit a **typewritten document**.
- For each exercise for which you develop Lisp functions you submit a printout of your documented functions. You may use comments in the file and hand draw diagrams on your listings. Use one file per exercise named `exercise-n.lsp`, where n corresponds to the exercise number. Please use the command enscript to print the `*.lsp` files.
  `enscript exercise-N.lsp`
- For each program for which you develop Prolog predicates hand in a printout of your documented predicates. You may use comments in the file and hand draw diagrams on your listings. Use one file per exercise named `exercise-n.pro`, where n corresponds to the exercise number. Please use the command enscript to print `*.pro` files.
- Before the deadline, submit a directory called report2 that should contain one `*.lsp` file for each exercise for which you wrote Lisp functions, and one `*.pro` file for each exercise for which you wrote Prolog predicates. Use the following Prism command.
  `submit 3401 r2 report2`

While you can develop your programs on your personal computer, be sure your files will load and execute correctly on Prism.

Question 1.

(2 points).

For this exercise all support functions are to embedded as anonymous lambda functions within a single definition. I found maplist to be useful for this exercise.

1. Write the functional **pair-combinations-from(list)** that returns a list of all combinations of two items from the **list** such that:
   $Result = \{(list[i], list[j]) \mid i \leq j \leq length(list)\}$.

   For example, consider the following.

   List = (a b c d)
   Result = ((a a)(a b)(a c)(a d)(b b)(b c)(b d)(c c)(c d)(d d))

2. Write the functional triple-combinations-from(list) that returns a list of all combinations of three items from the list, such that:
   $Result = \{(list[i], list[j], list[k]) \mid i \leq j \leq k \leq length(list)\}$.

   For example, consider the following.

   List = (a b c)
   Result = ((a a a)(a a b)(a a c)(a b b)(a b c)(a c c)(b b b)(b b c)(b c c)(c c c))

Question 2.

(1 Point).

The function **range** in the file **functionals.lsp** fails in Clisp on Prism when attempting to evaluate **(range 1 100,000)** commas are written to make it easier to see the hundred thousand) because recursion goes too deep. Write a recursive function **big-range(first last)** that uses range to produce subsequences and successfully evaluates **(length (big-range 1 10,000,000))** I doubt you want to see the list! Consider how to test your function to verify that your subsequences, when combined do not miss and/or duplicate integers.

Question 3.

(1 Point).

Write a version, **curry-v2** of the macro **curry** (see functionals.lsp) to use lambda functions in place of the **let** expressions.

Question 4.

(2 Points).

Write a Lisp macro `mycase` that translates the following macro call. Assume the input will be error free. The input lists can be any length.

```
(mycase (C1 C2 ...  Cn) (P1 P2 ...  Pn))
```
translates to the following:
```
(mycond (C1 P1) (C2 P2) ...  (Cn Pn))
```

- Variation 1: **mycase-v1** is to use recursion
- Variation 2: **maycase-v2** is to not to use recursion

Question 5.

(2 Points).

For this exercise the structure of the list contains sufficient information so there is no need for explicit counting. Do not use any built-in predicates that determine list length or list parity.

1. Define the Prolog predicate **middle(List, Mid)** that asserts that **Mid** is the middle item of an odd length list **List**. **Middle** is false if **List** is of even length. The predicate **append** is useful.

2. Define the Prolog predicate **middles(List, Middles)** that asserts that **Middles** is the list of middle elements, as defined above, for each List at the top level of **List**. **List** could contain any items at the top level. Your definition should give the correct list for **Middles** as the

first answer but will probably give multiple answers if you use ; in the query. Explain why you have multiple answers. Think about what you need to have available to prevent multiple answers from occurring.

Question 6.

(2 Points).

1. Define the Prolog predicate **arith_prog(TheList)** that asserts whether **TheList** is an arithmetic progression or not. You can assume **TheList** will only contain integers. An arithmetic progression is a sequence of the form $r, r + s, r + 2 * s, , r + n * s$, for some integer r and some positive integer s. For example $(2, 5, 8, 11)$ is an arithmetic progression with $r = 2, s = 3$ and $n = 3$. $\#theList \leq 2$ implies that $theList$ is an arithmetic progression. This exercise does not require counting or length predicates.

2. Define the Prolog predicate **minMaxMean(TheList, ArithmeticMean)** that asserts that **ArithmeticMean** is the arithmetic mean of the smallest and largest numbers in **TheList**. Assume that the list contains only numbers. An empty list has the atom **nil** as the arithmetic mean, to indicate that none exists. Your solution should only make one pass over the list.

Question 7.

(2 Points).

1. Write a Prolog predicate **substitute(OldValue, NewValue, List, Result)** that asserts **Result** is the result of substituting every occurrence of **OldValue** (at all levels) with **NewValue** in the list **List**.

2. Write a Prolog predicate **swap (ItemOne, ItemTwo, TheList, TheResult)** that asserts that **TheResult** is **TheList** with **ItemOne** and **ItemTwo** swapped if they are adjacent, in that order in **TheList**. Swapping takes place at all levels of **TheList**.