

CS3342 – Assignment 4

due Apr. 7, 2022

2-day no-penalty extension until: Apr. 9, 11:55pm
(SRA's cannot be used to extend the due date further)

1. (15pt) Consider the following Prolog rule:

$a(X, Y) :- b(Z), c(X, Z), d(X, Y, Z), e(Z, W).$

Rewrite this rule in predicate calculus, using:

- (a) four universal quantifiers; $\forall x \forall y \forall z \forall w$
- (b) two universal quantifiers and two existential quantifiers;
- (c) four existential quantifiers.

Solution:

$$\begin{aligned}
 & \textcircled{Q_1} \quad a(x, y) :- b(z), c(x, z), d(x, y, z), e(z, w). \\
 & \textcircled{a} \quad \forall x \forall y \forall z \forall w (a(x, y) \vee \neg b(z) \vee \neg c(x, z) \vee \neg d(x, y, z) \vee \neg e(z, w)) \equiv \\
 & \quad \forall x \forall y (a(x, y) \vee \forall z (\neg b(z) \vee \neg c(x, z) \vee \neg d(x, y, z) \vee \forall w (\neg e(z, w)))) \equiv \\
 & \quad \forall x \forall y (a(x, y) \vee \forall z (\neg b(z) \vee \neg c(x, z) \vee \neg d(x, y, z) \vee \neg \exists w (e(z, w)))) \equiv \\
 & \quad \forall x \forall y (a(x, y) \vee \forall z (\neg (b(z) \wedge c(x, z) \wedge d(x, y, z) \wedge \exists w (e(z, w))))) \equiv \\
 & \textcircled{b} \quad \forall x \forall y (a(x, y) \vee \neg \exists z (b(z) \wedge c(x, z) \wedge d(x, y, z) \wedge \exists w (e(z, w)))) \equiv \\
 & \quad \forall x \forall y (\neg \neg a(x, y) \vee \neg \exists z (b(z) \wedge c(x, z) \wedge d(x, y, z) \wedge \exists w (e(z, w)))) \equiv \\
 & \quad \forall x \forall y (\neg (\neg a(x, y) \wedge \exists z (b(z) \wedge c(x, z) \wedge d(x, y, z) \wedge \exists w (e(z, w))))) \equiv \\
 & \quad \forall x (\neg \exists y (\neg a(x, y) \wedge \exists z (b(z) \wedge c(x, z) \wedge d(x, y, z) \wedge \exists w (e(z, w))))) \equiv \\
 & \textcircled{c} \quad \neg \exists x \exists y (\neg a(x, y) \wedge \exists z (b(z) \wedge c(x, z) \wedge d(x, y, z) \wedge \exists w (e(z, w))))
 \end{aligned}$$

2. (25pt) Implement a sorting algorithm (of your choice) in Prolog as a predicate `my_sort(X, Sorted_X)`, working as shown below:

```
?- my_sort([1], X).
X = [1] .
?- my_sort([2,1,3], X).
X = [1, 2, 3] .
```

Submit your code as a file `my_sort.pl`.

Solution:

```
insert(X, [], [X]).
insert(X, [Y|T], [X, Y|T]) :- X <= Y.
insert(X, [Y|T], [Y|NT]) :- X > Y, insert(X, T, NT).

insert_sort([], L, L).
insert_sort([H|T], L, SL) :- insert(H, L, NL), insert_sort(T, NL, SL).

my_sort(L, SL) :- insert_sort(L, [], SL).
```

3. (60pt) Implement the following Scheme functions:

```
(define my-permutations
  (lambda (L) ...
    (define list-lex-less?
      (lambda (L1 L2) ...
        (define my-sort
          (lambda (L comp-pred?) ...
```

`my-permutations` generates a list of all permutations of a given list (in any order); `list-lex-less?` compares two lists and outputs true iff the first is smaller than the second in lexicographic order; `my-sort` sorts a given list (with an algorithm of your choice) using the given `comp-pred?` predicate to compare elements. The three functions are then put together into `sorted-permutations` to produce all permutations of a given list in lexicographical order.

```
(define sorted-permutations
  (lambda (L) ...
```

Here are some examples:

```
(my-permutations '(1 3 2)) => '((1 3 2) (3 1 2) (3 2 1) (1 2 3) (2 1 3) (2 3 1))
(list-lex-less? '(1 2) '(1 2)) => #t
(list-lex-less? '(1 2) '(1 2 3)) => #t
(list-lex-less? '(2) '(1 3)) => #f
(list-lex-less? '(1 2 3) '(1 2)) => #f
(my-sort '(2 3 1) <) => '(1 2 3)
(my-sort '((3) (1 2 3) (1 3)) list-lex-less?) => '((1 2 3) (1 3) (3))
(sorted-permutations '(1 3 2)) => '((1 2 3) (1 3 2) (2 1 3) (2 3 1) (3 1 2) (3 2 1))
```

You are required to provide pure functional implementations from scratch, that do not employ advanced functions or imperative features. Therefore, you are allowed to use *only* the following basic Scheme functional constructs:

– function creation: `lambda`

- binding: define, let, let*, letrec
- booleans: not, and, or
- conditionals: if, cond
- basic list operations: car, cdr, cons, list, append, null?
- mapping: map, apply

Submit your code as a file `sorted-permutations.rkt`.

Solution:

```
(define my-permutations
  (lambda (L)
    (let
      ((insert-all
        (lambda (e Ls)
          (let
            ((insert-one
              (lambda (L)
                (letrec
                  ((helper
                     (lambda (L R)
                       (if (null? R)
                           (list (append L (list e)))
                           (append (list (append L (list e) R))
                                   (helper
                                      (append L (list (car R)) (cdr R)))))))
                  (helper '() L))))))
          (apply append (map insert-one Ls))))))
      (cond ((null? L) '())
            ((null? (cdr L)) (list L))
            (else (insert-all (car L) (my-permutations (cdr L)))))))

(define list-lex-less?
  (lambda (L1 L2)
    (cond
      ((null? L1) #t)
      ((null? L2) #f)
      ((< (car L1) (car L2)) #t)
      ((> (car L1) (car L2)) #f)
      (else (list-lex-less? (cdr L1) (cdr L2)))))

(define my-sort
  (lambda (L comp-pred?)
    (letrec ((partition
              (lambda (e L L1 L2)
                (if (null? L) (cons L1 L2)
                    (let ((c (car L)))
                      (if (comp-pred? c e)
                          (partition e (cdr L) (cons c L1) L2)
                          (partition e (cdr L) L1 (cons c L2)))))))
      (cond
        ((null? L) L)
        ((null? (cdr L)) L)
```

```

      (else (let* ((L12 (partition (car L) (cdr L) '() '()))
                  (L1 (car L12))
                  (L2 (cdr L12)))
              (append (my-sort L1 comp-pred?)
                      (list (car L))
                      (my-sort L2 comp-pred?))))))

(define sorted-permutations
  (lambda (L)
    (my-sort (my-permutations L) list-lex-less?)))

```

READ ME! Submit your answers as a *single pdf file* in OWL. Solutions should be typed but readable (by others!) hand-written solutions are acceptable. Source code, if required, is submitted as separate files.

L^AT_EX: For those interested, the best program for scientific writing is L^AT_EX. It is far superior to all the other programs, it is free, and you can start using it in minutes; here is an introduction: <https://tobi.oetiker.ch/lshort/lshort.pdf>