

# CS 476 Homework #6 Due 10:45am on 3/10

**Note:** Answers to the exercises listed below and the code solution for Exercise 2 should be emailed in *typewritten form* (latex formatting preferred) by the deadline mentioned above to `abir2@illinois.edu`.

1. Solve Exercise 93 in *STACS*, pg. 93.

**Hint.** It makes no sense for you to even try to solve this exercise if you have not first thought carefully about the meaning of an unsorted  $\Sigma$ -algebra  $\mathcal{A}$  as a pair  $\mathcal{A} = (A, \_A)$  with  $A$  a set, and  $\_A$  an interpretation function.

2. (Exercise suggested by Peter Ölveczky). Do the following:

- Define a data type `ListQid` of lists of `Qid`-elements. In this exercise you will use both lists and sets, and you are therefore advised to use a symbol other than `_ _` for list concatenation. Use e.g., `_:_` or `_ ; _` or whatever your favorite list concatenation operator is.
- Define a data type `Set-ListQid` of sets of lists of quoted identifiers. Then define only those functions needed to solve the next part of this exercise.
- Define a function

```
op perm : ListQid -> Set-ListQid .
```

which takes a list of `Qids` and returns the set of all permutations of this list. (A permutation of a list is a list where the elements are the same but are rearranged.) For instance, the set of all permutations of the list `'a : 'b : 'c` (using `_:_` as the list concatenation operator) is the set (using `_ _` as set union operator)

```
('a : 'b : 'c) ('a : 'c : 'b) ('b : 'a : 'c)
('b : 'c : 'a) ('c : 'a : 'b) ('c : 'b : 'a)
```

**Hint:** This exercise is somewhat harder than previous exercises. One idea to generate all permutations of a list such as `'a : 'b : 'c` is to generate `'a` plus all permutations of `'b : 'c` and `'b` plus all permutations of `'a : 'c` and `'c` plus all permutations of `'a : 'b`. That is, we gradually construct each permutation.

This suggests that the job of actually generating all permutations could be done by an auxiliary function

```
op p : ListQid ListQid ListQid -> Set-ListQid .
```

where `p(L1, L2, L3)` generates the set of all permutations of `L1 : L2 : L3` that begin with `L1` followed by a permutation of `L2 : L3`. It will do so by generating lists which start with `L1`, followed by an element chosen from `L2`, followed by a permutation of `L3` concatenated with the remaining elements of `L2`. But `p` may also choose *not* to pick a given element of `L2` as the next element after `L1`. So you need to think carefully about how such a function `p` will work, and how to use the function's third argument in recursive calls to `p`.

Last, but not least, you can achieve a very simple and elegant solution of this problem by taking full advantage of suitable equational axioms like `assoc`, `comm`, and `id`: when defining lists and sets. Using these axioms, it is possible to solve the above problem with just five equations! This is again another good example of the motto:

*Declarative Programming = Mathematical Modeling*

Obviously, you are not required to define the above auxiliary function `p`. You can give a different solution of your own to solve this problem.