**CSCI 301, Summer 2019**

# Lab 3 (Racket Programming: Recursion and Set Operations)

**DUE: 11:59pm Monday, 7/22, Online submission**

**30 Points Total**

- This is an **individual** assignment. Work through the following lab.
- In this lab assignment, you will work experimentally with the DrRacket language on **Recursions and Set Operations**.
- Keep in mind that in addition to this lab, there are Racket and Scheme resources linked in the syllabus if you need help.

In Racket sets can be represented as lists. However, unlike lists, the order of values in a set is not significant. Thus both `(1 (2 3))` and `((3 2) 1))` represent the same set.

In lab 2, we have assumed that a set contains only atomic values (numbers, string, symbols, etc.) and it does not contain duplicate members. However, In general sets can contain other sets (here you may again assume that a set does not contain duplicate members). See the examples in the exercises below. Extend your solutions to lab 2 to allow sets to contain other sets. Make sure to test additional cases of sets containing sets containing sets, etc.

**Hint:** You may write helper functions and call them in the primary functions. A useful helper function could be to test whether a given sublist/set exists in the given primary list/set.

1. Write a Racket function `(set-equal? L1 L2)` that tests whether L1 and L2 are equal. Two sets are equal if they contain exactly the same members, ignoring ordering (or in other words, two sets are equal if they are a subset of each other). For example

```
(set-equal? '(1 (2 3)) '((3 2) 1)) ---> #t
(set-equal? '(1 2 3) '((3 2)1)) ---> #f
(set-equal? '(1 2 3) '((1 2 3))) ---> #f
```

2. Two common operations on sets are **union** and **intersection**. The union of two sets is the set of all elements that appear in either set (with no repetitions). The intersection of two sets is the set of elements that appear in both sets.
   Write Racket functions `(union S1 S2)` and `(intersect S1 S2)` that implement set union and set intersection. For example

```
(union '(1 (2) 3) '(3 2 1)) ---> (1 2 3 (2))
(union '((1 2 3)) '((3 4 5))) ---> ((1 2 3) (3 4 5))
(union '((1 2 3)) '((3 2 1))) ---> ((1 2 3))
(intersect '((1 2 3)) '((3 2 1))) ---> ((1 2 3))
(intersect '((1 2 3)) '((4 5 6))) ---> ()
(intersect '((1) (2) (3)) '((2) (3) (4))) ---> ((2) (3))
```

   The ordering of the elements in your answer may differ from the above.

**You must use recursion, and not iteration. You may not use side-effects (e.g. set!).**

The solutions will be turned in by posting a single Racket program (lab03. rkt) containing a definition of all the functions specified.