1 CSE 341 — Winter 2003 — Scheme Programming Assignment

Due Feb 21 in class.

Write and test a set of Scheme functions to perform simplification of symbolic expressions involving sets. These Scheme functions must all be written in a pure functional style (no side effects). You just need to handle sets of integers, which can be represented in Scheme as lists of integers. The order of the integers in the list isn't significant, but there shouldn't be any duplicates.

First, write and test a predicate `good-set?` that tests whether its argument is a properly represented set, i.e. it is a list consisting only of integers, with no duplicates. For example:

```
(good-set? '(1 5 2))  => #t
(good-set? ())  => #t
(good-set? '(1 5 5))  => #f
(good-set? '(1 (5) 2))  => #f
```

You can run `good-set?` on sample input, and all your other functions can assume that sets are properly represented.

Then write and test functions to perform set union, intersection, and set difference.

Next, write and test a function `set-simplify` that does simplification of symbolic expressions involving sets. Your function should perform the following simplifications. In the rules below, x, y, and z are arbitrary expressions involving the set operations $\cup$, $\cap$, and $\setminus$ (set union, intersection, and difference respectively). These rules are given in standard set notation, but your Scheme functions should manipulate expressions using Scheme's expression syntax.

$$\begin{aligned} x \cup \emptyset &\Rightarrow x \\ x \cap \emptyset &\Rightarrow \emptyset \\ x \cup x &\Rightarrow x \\ x \cap x &\Rightarrow x \\ \emptyset \setminus x &\Rightarrow \emptyset \\ x \setminus \emptyset &\Rightarrow x \\ x \setminus x &\Rightarrow \emptyset \end{aligned}$$

Also include the commutative version of the rules as appropriate (e.g. a rule for $\emptyset \cup x$).

Finally, if you have an expression involving actual sets, rather than variables, simplify the expression by evaluating the expression. (Use the union, intersection, and set difference functions you defined earlier.) For example:

$$\{1, 5, 2\} \cup \{5, 10\} \quad \Rightarrow \quad \{1, 5, 2, 10\}$$

Here are some example calls to `set-simplify`:

```
(set-simplify '(union x x))  =>  x
(set-simplify '(intersect x ()))  =>  ()
(set-simplify '(difference x x))  =>  ()
(set-simplify '(union '(1 5 2) '(5 10)))  => '(1 5 2 10)
(set-simplify '(union '(1 5 2) x))  => (union '(1 5 2) x)
(set-simplify ''(1 5 2))  => '(1 5 2)
```

Note that `x` in the argument is a symbolic variable, while `'(1 5 2)` represents a constant, namely the set $\{1, 5, 2\}$. Notice also that in the final two cases, no simplification was possible. Finally, in the last example, there are two quotes in a row — this example could also have been written

```
(set-simplify '(quote (1 5 2)))  => '(1 5 2)
```

In standard mathematical notation, the above calls are equivalent to:

$$\begin{aligned} x \cup x &\Rightarrow x \\ x \cap \emptyset &\Rightarrow \emptyset \\ x \setminus x &\Rightarrow \emptyset \\ \{1,5,2\} \cup \{5,10\} &\Rightarrow \{1,5,2,10\} \\ \{1,5,2\} \cup x &\Rightarrow \{1,5,2\} \cup x \\ \{1,5,2\} &\Rightarrow \{1,5,2\} \end{aligned}$$

Here are some additional, more complex, examples to try (again, in mathematical notation):

$$\begin{aligned} x \cap y \cap (\{1,3\} \setminus \{5,8,1,10,3\}) &\Rightarrow \emptyset \\ (a \cup (b \setminus b)) \cap a &\Rightarrow a \\ ((((\{0,10,3\} \cup \{10,3,100\}) \cap \{100,50,3\}) \setminus \{3\}) \cup x &\Rightarrow \{100\} \cup x \end{aligned}$$

You can also check that your simplifications are working correctly by giving the variables values and using `eval` on the original and simplified expressions. For example:

```
(define x '(10 15 20))

(set-simplify '(union x x))  =>  x
(set-simplify '(intersect x ()))  =>  ()
(set-simplify '(difference x x))  =>  ()

(eval '(union x x) user-initial-environment)  =>  (10 15 20)
(eval (set-simplify '(union x x)) user-initial-environment)  =>  (10 15 20)
```

(Make sure you understand why this works.)

2 Testing Your Functions

You should systematically test your top-level functions (`good-set?`, `set-simplify`, `union`, `intersect`, and `difference`), and show the output from the tests. For example, for `set-simplify`, run it on the test cases shown above, and any others you think are necessary. (For example, you should simplify the empty set also.) Similarly, test `union`, `intersect`, and `difference` with empty and non-empty sets. In addition, include some tests using `eval`, as above.

There isn't an analog to JUnit for Scheme, but it's still convenient to write a `test-sets` function that will run tests on all your functions, so that it's easy to perform the tests repeatedly. The `test-sets` function should print out some informative messages about each test, and then return the final result (`#t` if all the functions passed all of the tests, and `#f` otherwise).

3 Hints

We're representing set *expressions* as list structures. Inside an expression, a set *constant* is written by quoting the list that represents the set. When Scheme reads in a list containing a quoted list, the tick mark (`'`) is converted to `quote`. For example, if you type in the expression `'(union '(1 2) '(3))`, this will print as (and be represented internally as):

```
(union (quote (1 2)) (quote (3)))
```

There was a previous CSE 341 Scheme project involving simplification of symbolic *arithmetic* expressions two years ago, available at:

`http://www.cs.washington.edu/education/courses/cse341/00sp/assignments/scheme-program.html`

There is a sample solution for that project on ceylon.cs.washington.edu on `~borning/scheme/simplify.scm`

You may want to copy this program to your own directory and experiment with it. (One thing that will be a bit more difficult in this present assignment is figuring out what's quoted and where . . . this wasn't such an issue in simplifying arithmetic expressions, since numbers don't need to be quoted — they evaluate to themselves.)

4 Turnin Directions

To turn in your work, please put all of your functions, using the names specified in the homework, in a single file called 'homework5.scm'. Make a separate text file with the output of your tests with clear headings for the output from the different functions. Use the following turnin command to turn the file in electronically:

```
turnin homework5.scm homework5.output.txt
```

Do not zip or tar the files before turning them in. To check that the turnin was successful, use:

```
turnin -v
```

See the 341 homework submission guidelines for general information, and hints on putting the test output into a file.

5 Extra Credit

For up to 10% extra credit, add additional simplification rules. The above rules are exhaustive if you just look at expressions involving two variables, or a variable and a constant, or two constants, but there are other simplifications if you consider other expressions. For example:

$$\begin{aligned} (x \cup y) \cap x &\Rightarrow x \\ (x \cup y) \setminus x &\Rightarrow y \setminus x \\ w \cup x \cup y \cup z \cup w &\Rightarrow w \cup x \cup y \cup z \end{aligned}$$

For full extra credit, rather than just adding a grab-bag of additional simplification rules, devise a comprehensive approach to the issue — what are the classes of simplification rules you are considering? How did you pick the ones you included? You might want to look at a textbook on set theory or finite mathematics for ideas. (I actually don't know what the answer is to this part of the question, so I'm looking forward to seeing what some of you come up with.)