CS 301: Formal Languages and Functional Programming, Fall, 2016
Programming Assignment 1, due: Oct 26, 23:59

1. Implement a Scheme function called iff, which takes two truth values as input and returns true (`#t`) when the input values bear the "if and only if" relation to one another and false (`#f`) otherwise.

   For example, `(iff #f #f)` returns `#t`.

2. Implement a Scheme function called exclusive-or, which takes two truth values as input and returns true if and only if they bear the exclusive-or relation to one another.

3. Implement a Scheme function called difference, which takes two lists interpreted as sets and returns the list representing the difference, that is, the first input set minus the second.

   For example, `(difference '(a b c d) '(a c e))` returns `'(b d)`.

4. Implement a Scheme function called symmetric-difference. It takes as input two lists interpreted as sets and returns their symmetric difference.

   For example, `(symmetric-difference '(a b c) '(b c d e))` returns `'(a d e)`.

5. Implement a Scheme function stem, a function that returns the list obtained by removing the last element from the argument list. Do not use the built-in function reverse.

6. Implement a Scheme function, replicate, that takes as arguments an stom and a list of atoms. The return list is the input list with each instance of the atom replicated. As a boundary condition, if the atom is not in the input list, the function returns that input list.

   For example, `(replicate fi (fee fi fo fum fi))` returns `(fee fi fi fo fum fi fi)`.

7. Binary counter: Implement a Scheme function next, a function that returns the next binary number by incrementing the given binary number. For example,

   - `(next '(0 0 1))` returns `'(0 1 0)`
   - `(next '(1 1 0))` returns `'(1 1 1)`
   - `(next '(1 1 1 1))` returns `'(0 0 0 0)`
   - `(next '(1 0 1 1 1)` returns `'(1 1 0 0 0)`

   and so forth.

   Hint: It will be easier to implement the counter if the least significant bit is the list head instead of the last element. Call this simpler version next1. Then, you can use the reverse function that you learned in the Little Schemer to obtain next as

```
(define next
 (lambda (x)
  (reverse (next1 (reverse x)))))
```