CSE 3302 Lab Assignment 5

Due April 23, 2013

Goal:

Understanding of Scheme and scope concepts.

Requirements:

1. Write a Scheme program to evaluate a quantified boolean expression (formula):

   a. The input expression will be a nested S-expression consisting of the following sub-expressions:

      1. All syntactic elements from lab 2 (N, O, A, and names in functional form).
      2. Implication of the form (I *p-exp q-exp*), interpreted as $p-exp \supset q-exp$ i.e. $\overline{p-exp} \vee q-exp$.
      3. Equivalence (iff) of the form (E *p-exp q-exp*), interpreted as $p-exp \leftrightarrow q-exp$.
      4. Universal quantification of the form (F *prop s-exp*). This expression is true only when *s-exp* evaluates to true in both of these situations:

         a. True is substituted for all unbound occurences of *prop* within *s-exp*.
         b. False is substituted for all unbound occurences of *prop* within *s-exp*.

      5. Existential quantification of the form (T *prop s-exp*). This expression is true only when *s-exp* evaluates to true in at least one of these situations:

         a. True is substituted for all unbound occurences of *prop* within *s-exp*.
         b. False is substituted for all unbound occurences of *prop* within *s-exp*.

   b. The final output for an expression is simply `#t` or `#f`.

   c. Due to the quantifiers and scoping, this lab does not have the (simple) notion of a truth assignment like lab 2.

2. Email your program to `mehra.nourozborazjany@mavs.uta.edu` by 10:45 a.m. on April 23, 2013.

Getting Started:

1. Enlightment regarding this topic may be obtained from the following resources:

   H. Chen. "A Rendezvous of Logic, Complexity, and Algebra", *ACM Computing Surveys*, Vol. 42, No. 1, Article 2, December 2009. (`http://dl.acm.org/citation.cfm?doid=1592451.1592453`)

   C.H. Papadimitriou. *Computational Complexity*, Addison-Wesley, 1993. ISBN 0201530821.

2. My solution for lab 2 is on the course webpage.

3. Wirth's "stepwise refinement" is highly recommended . . .

```
(define (process exp)
  (begin
    (displayln exp)
    (if (qbf? exp)            ; check syntax
        (if (allBound? exp)   ; check scoping
            (qbfEval exp '()) ; search by expanding quantifiers
            (displayln "unbounded name"))
        (displayln "malformed expression"))))
```

4. Unlike lab 2, there is no “collecting”.

5. Do not confuse this assignment with the more customary predicate calculus, where quantification is over individual variables.

6. The Ten Commandments and The Five Rules from *The Little Schemer* will lead you to many days of happiness.

7. `set!` will lead to nights of suffering (and loss of points).