In-Class Problems — Week 2, Mon

Problem 1. Two Boolean formulas $F_1(x_1, \ldots, x_n)$ and $F_2(x_1, \ldots, x_n)$ are equivalent iff they yield the same truth value for all truth assignments to the variables $x_1, \ldots, x_n$.

(a) Describe an infinite set of equivalent Boolean formulas.

(b) How many equivalence classes are there of formulas with (at most) variables $x_1, \ldots, x_n$?

Problem 2. A Scheme expression satisfies the “Variable Convention” if no variable identifier is bound more than once, and no identifier has both bound and unbound occurrences. For example, the expression

\[
\text{(let ((x 2) (y 5))}
\quad (+ ((\text{lambda} (x) (+ x 1)) 3) ((\text{lambda} (z) (+ x y z 11)) 99) z)).
\]

violates the Variable Convention because $x$ is bound twice—once by let and once by lambda, and also because $z$ has both a bound and an unbound occurrence.

Any expression can be slightly modified to satisfy the Convention solely by adding integer suffixes to some of the bound identifiers—in a way that preserves all the binding structure and all the computational behavior of the original expression.

For example, by adding suffix 0 to the $x$’s and $z$’s bound by the lambda’s, we obtain an equivalent expression which satisfies the Variable Convention:

\[
\text{(let ((x 2) (y 5))}
\quad (+ ((\text{lambda} (x0) (+ x0 1)) 3) ((\text{lambda} (z0) (+ x y z0 11)) 99) z)).
\]

Show how to add such suffixes to the identifiers in

\[
(a b c d e)
\quad (\text{let ((a e) (b c))}
\quad (a b c d e)
\quad (\text{letrec ((a c) (c b))}
\quad (a b c d e)))))))
\]
to obtain an equivalent expression satisfying the Variable Convention. (See the Scheme reference manual to find out the scoping rules for letrec.)

Problem 3. (a) Define a Scheme procedure `self-compose` which, given a one-parameter procedure argument, `f`, returns a procedure that computes \((f \circ f)\), that is, the composition of `f` with itself. For example, the Scheme expressions

```
(define (self-compose f) <your definition>)
(define (s n)(* n n))
((self-compose s) 3)
```

would return the integer 81.

(b) What should \(((self-compose self-compose) s) 3)\) return? Explain.

Problem 4. Define a Scheme procedure `abc-strings` which applied to any positive integer argument, `n`, will print out all the strings of length `n` over the alphabet \(\{a, b, c\}\) in alphabetical order.