CS 2800 Section 1 – Exam 2 – Spring 2012

Name: _____

Student Id (last 4 digits): _____

- You must take the exam in the section you are registered for.
- Write down the answers in the space provided.
- Please write clearly. If we can't read what you write, we can't give you credit for it.
- You may use anything we covered in class or in the class notes. Everything else needs to be defined. If you have any questions, please ask!

Question	Points	out of
1		240
2		420
3		540
Total		1200

Good luck!

Question 1. (240 = 50 + 190 points)

(a) Given the formula φ and the substitution σ below, determine $\varphi|_\sigma$, i.e. the result of applying σ to φ.

φ = `(equal (+ (len alpha) (len beta) (len gamma))`
`(len (cons 'alpha (beta gamma))))`

σ = `((alpha '(0 1)) (beta '(2)) (gamma '(3 4)))`

$\varphi|_\sigma$ =

(b) Among the following three expressions f_1, f_2, f_3, identify all pairs (f_i, f_j) such that $i \neq j$ and there exists a substitution σ such that $f_i|_\sigma = f_j$. For each pair, if σ exists, specify it. Otherwise, write "none".

f_1: `(h (cons a y) (list y '(2 3)) (/ (abs (fib x )) r))`

f_2: `(h (cons 'a z) (list z '(2 3)) (/ (abs (fib r )) x))`

f_3: `(h (cons 'a y) (list y '(2 3)) (/ (abs (fib x)) x))`

- σ such that $f_1|_\sigma = f_2$:
- σ such that $f_2|_\sigma = f_1$:
- σ such that $f_1|_\sigma = f_3$:
- σ such that $f_3|_\sigma = f_1$:
- σ such that $f_2|_\sigma = f_3$:
- σ such that $f_3|_\sigma = f_2$:

Question 2. (420 = 140 + 280 points)

(a) Albert is considering the following function:

```
(defunc wicked (a b c)
  :input-contract (and (integerp a) (posp b) (integerp c))
  :output-contract (integerp (wicked a b c))
  (cond ((>= a b) a)
        ((>= b c) b)
        ((< c 0)  (wicked (+ a b) b c))
        (t        (wicked a b (+ a b)))))
```

He claims that this function will be admitted by ACL2s since its body is a legal expression satisfying all body contracts, and the output contract is trivially satisfied as well. But he forgot something! Show that the function does not (always) terminate, by giving an input in the form of concrete values `a`, `b`, `c` that satisfies the input contract but causes an infinite execution.

(b) Albert admits there were **several** typos in his definition. His new proposal is (read carefully!):

```
(defunc wicked (a b c)
  :input-contract (and (posp a) (posp b) (integerp c))
  :output-contract (integerp (wicked a b c))
  (cond ((>= a b) a)
        ((< c 0)  (wicked (+ a b) b c))
        (t        (wicked a b (- a b)))))
```

Prove that this function terminates, by defining a measure function in the template below.

Hint: First run function `wicked` on a few well-chosen test cases (exercising all clauses of the `cond`) and see what happens. How long can chains of recursive calls to this function actually be?

```
(defunc m (a b c)
  :input-contract ...
  :output-contract ...
  ...
```

Be sure that `m` would be admitted by ACL2s. Then prove that, on every recursive call of `wicked`, the value of `m` decreases, under the conditions of that recursive call.

Question 3. (540 = 60 + 160 + 320 points)

Consider the following recursive definitions of natural-number addition and multiplication, and the following **four** theorems.

```
(defunc plus (a b)
  :input-contract (and (natp a) (natp b))
  :output-contract (natp (plus a b))
  (if (equal b 0)
    a
    (+ (plus a (- b 1)) 1)))

(defunc times (a b)
  :input-contract (and (natp a) (natp b))
  :output-contract (natp (times a b))
  (if (equal b 0)
    0
    (plus (times a (- b 1)) a)))

(defthm plus-commutative
  (implies (and (natp a) (natp b))
           (equal (plus a b)
                  (plus b a))))

(defthm plus-associative
  (implies (and (natp a) (natp b) (natp c))
           (equal (plus a (plus b c))
                  (plus (plus a b) c))))

(defthm plus-greater
  (implies (and (natp a)
                (natp b)
                (> a 0))
           (> (plus a b) 0)))

(defthm plus-minus-1
  (implies (and (natp a)
                (natp b)
                (> a 0))
           (equal (- (plus a b) 1)
                  (plus (- a 1) b))))
```

You may assume that the functions are admitted by ACL2s, and that the theorems have been proved. You may use the theorems in your proof below. If you do, quote the theorem you are using, and specify the substitution that you used to obtain an instance of the theorem. You may **not** apply any facts of natural arithmetic to `plus` and `times`, as we have neither proved, nor stated as a theorem, that these implement addition and multiplication.

The following conjecture comes up as part of an inductive proof that multiplication *distributes* over addition:

```
(implies (and (natp a)
              (natp b)
              (natp c)
              (> b 0)
              (implies (natp (- b 1))
                       (equal (times a (plus (- b 1) c))
                              (plus (times a (- b 1)) (times a c)))))
         (equal (times a (plus b c)) (plus (times a b) (times a c))))
```

(a) Extract the context from the conjecture.

(b) Determine the derived context. You should also add the following fact to your context: `(> (plus b c) 0)`. How do you justify it?

(c) Prove the conjecture by equational reasoning. To help you, the number of steps it may take and some intermediate results are provided (although your proof strategy may differ; these are just meant as guidelines). If you are stuck, look at the theorems in your arsenal, and at the (derived) context.

```
Proof:

  (times a (plus b c))

=



=



=

  (plus (plus (times a (- b 1)) (times a c)) a)

= { plus-commutative }



= { plus-associative }



= { plus-commutative }

  (plus (plus (times a (- b 1)) a) (times a c))

=

  (plus (times a b) (times a c))
```